Recursion

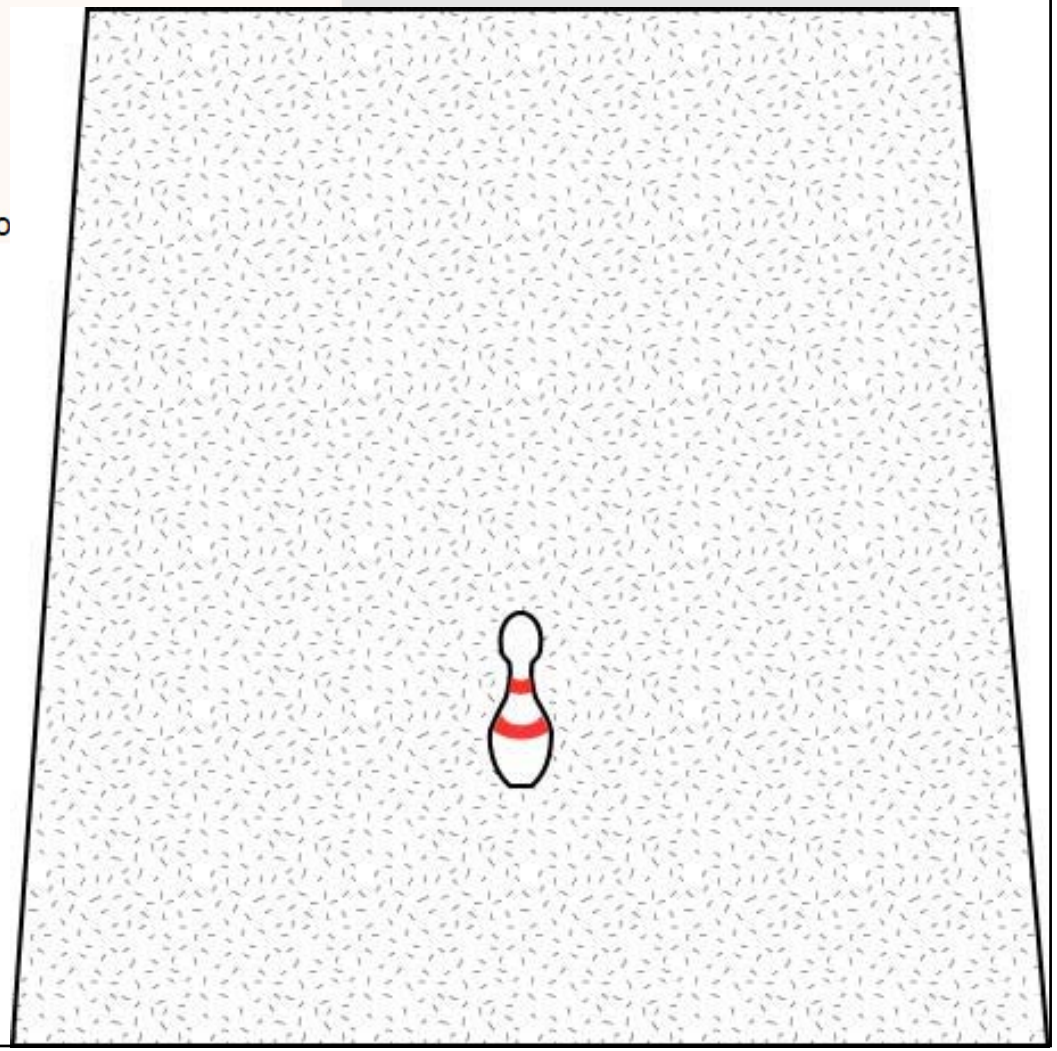
- Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfills the condition of recursion, we call this function a recursive function.

  A recursive function has to terminate to be used in a program. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a base case. A base case is a case, where the problem can be solved without further recursion. A recursion can lead to an infinite loop, if the base case is not met in the calls.

The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

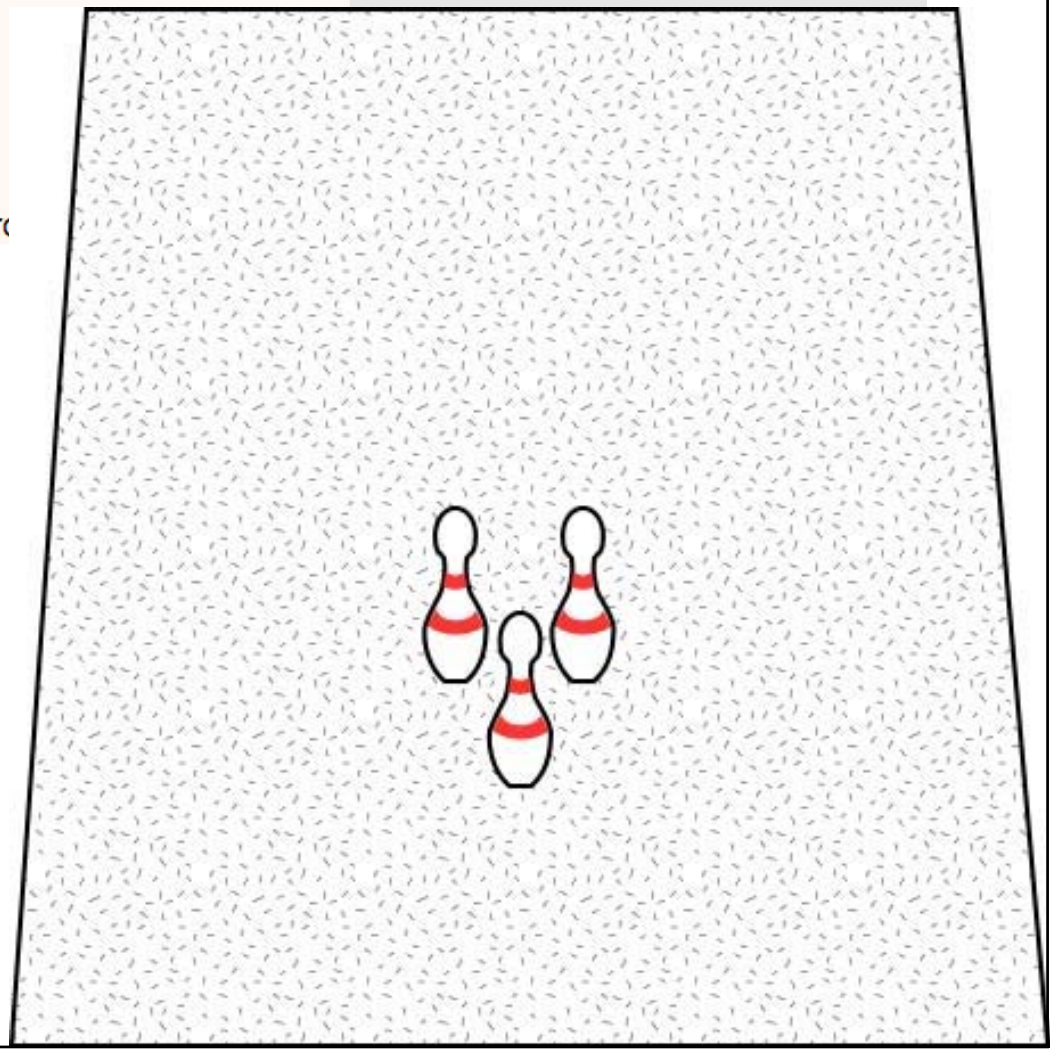
Click the image to see the added ro



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

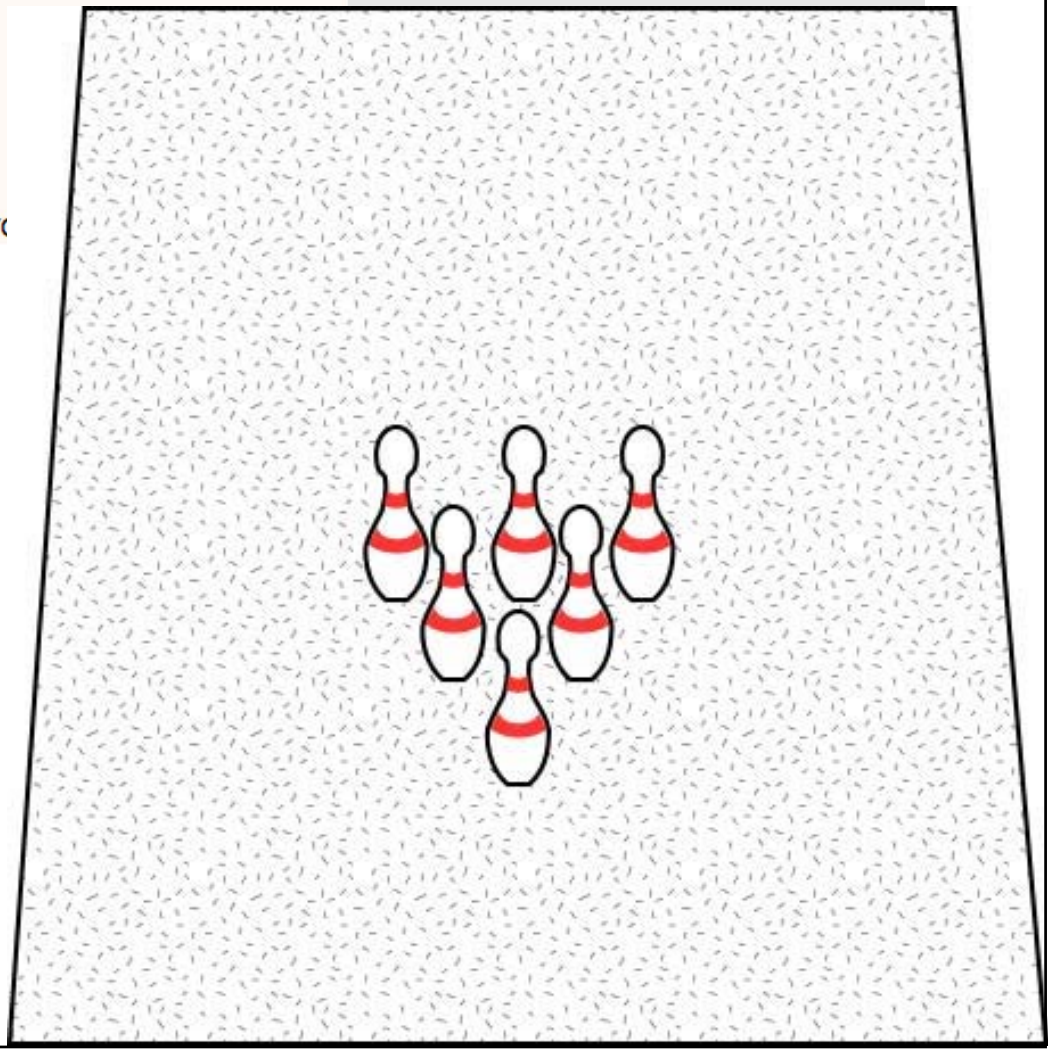
Click the image to see the added r



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

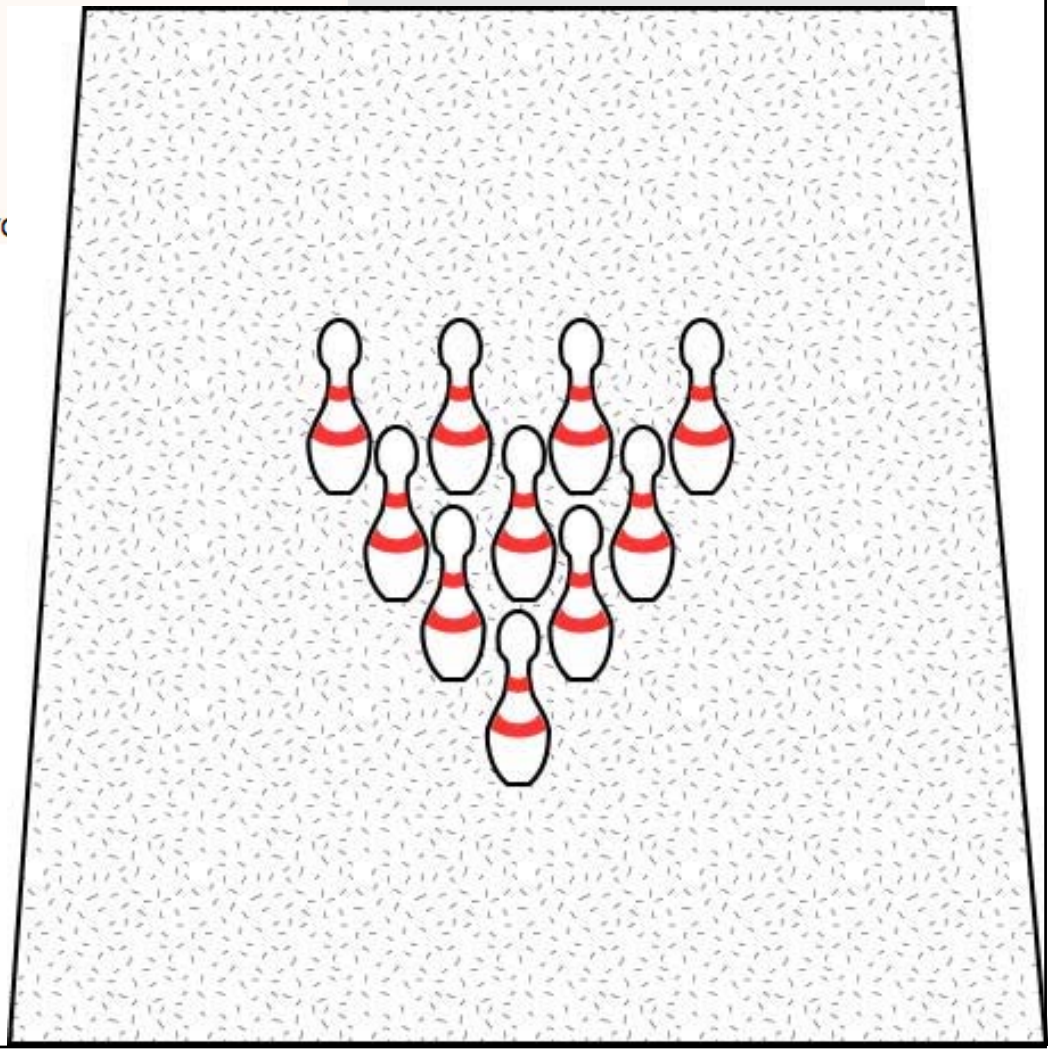
Click the image to see the added r



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

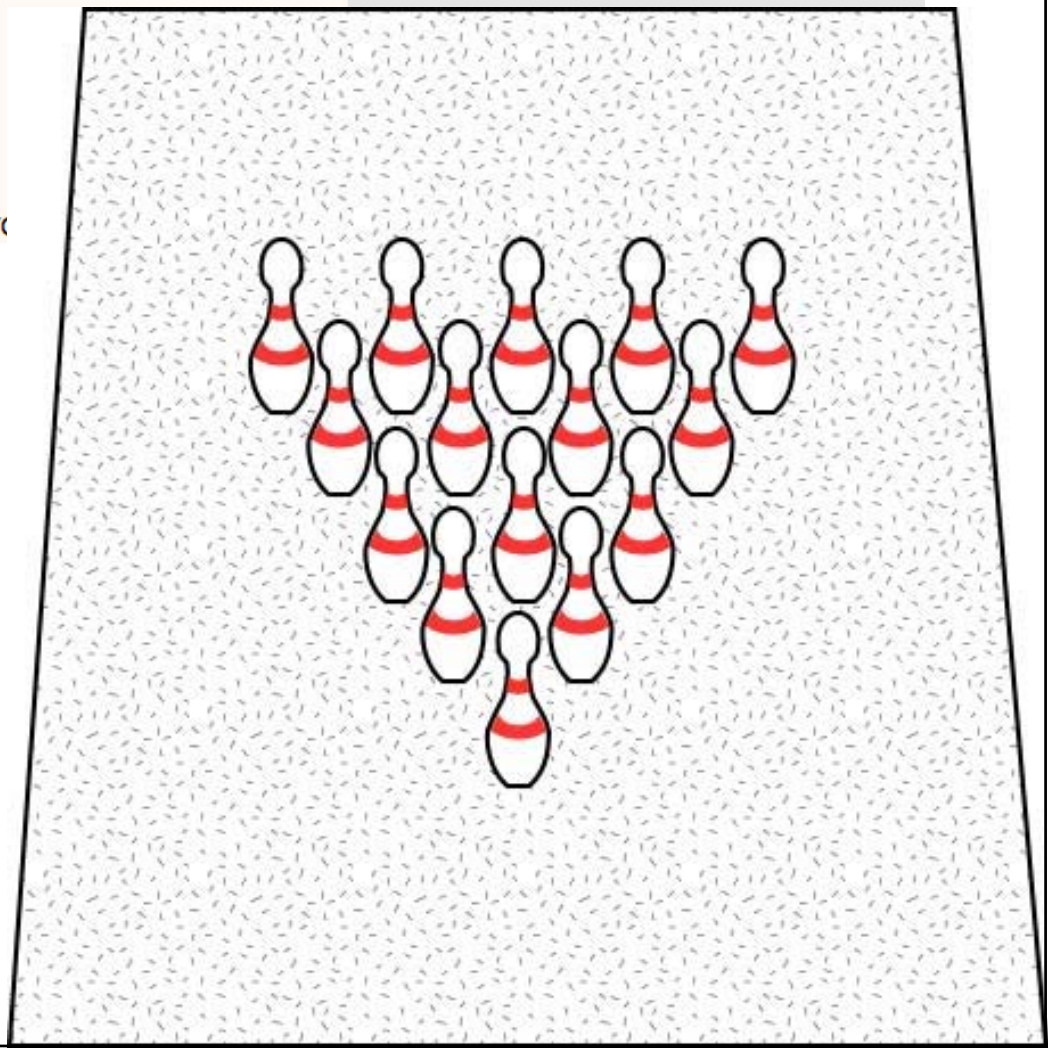
Click the image to see the added r



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

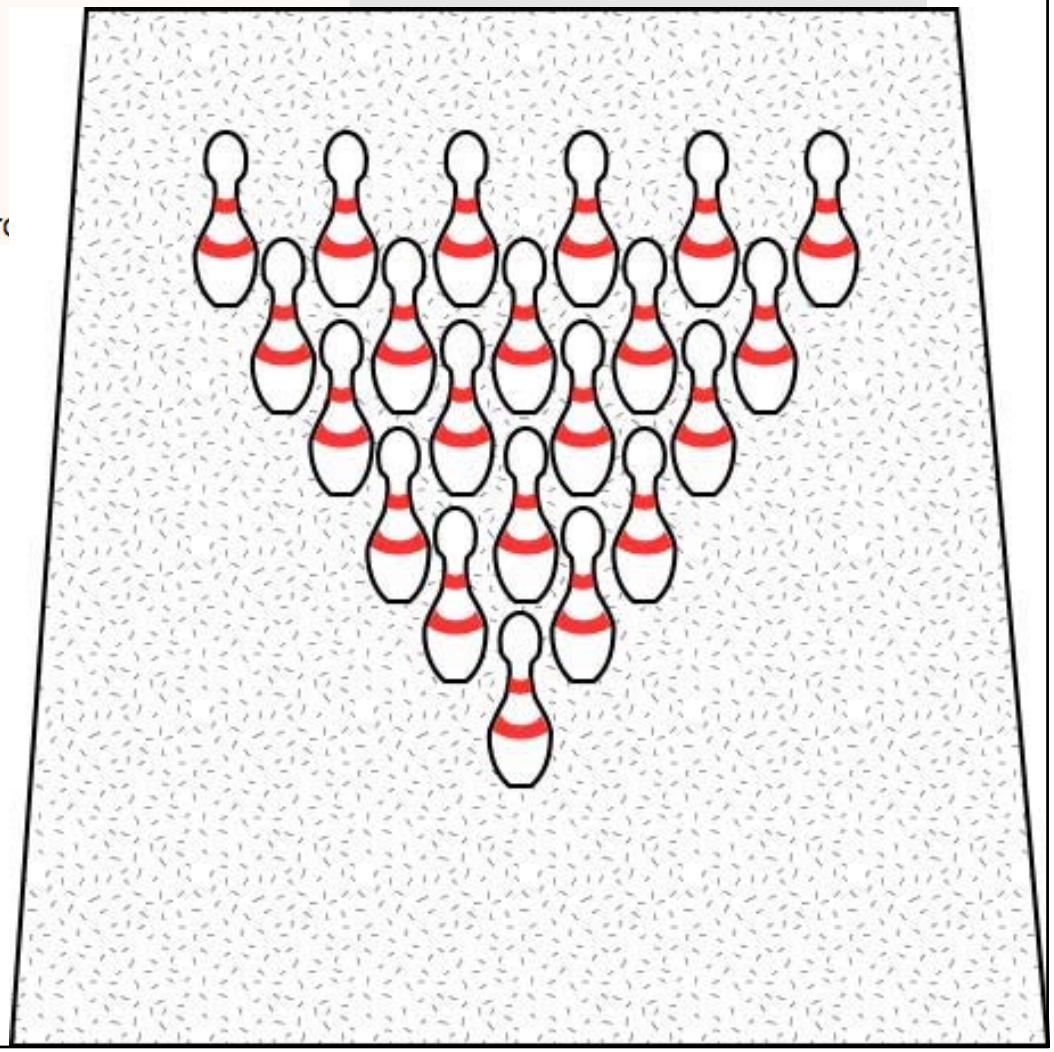
Click the image to see the added r



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

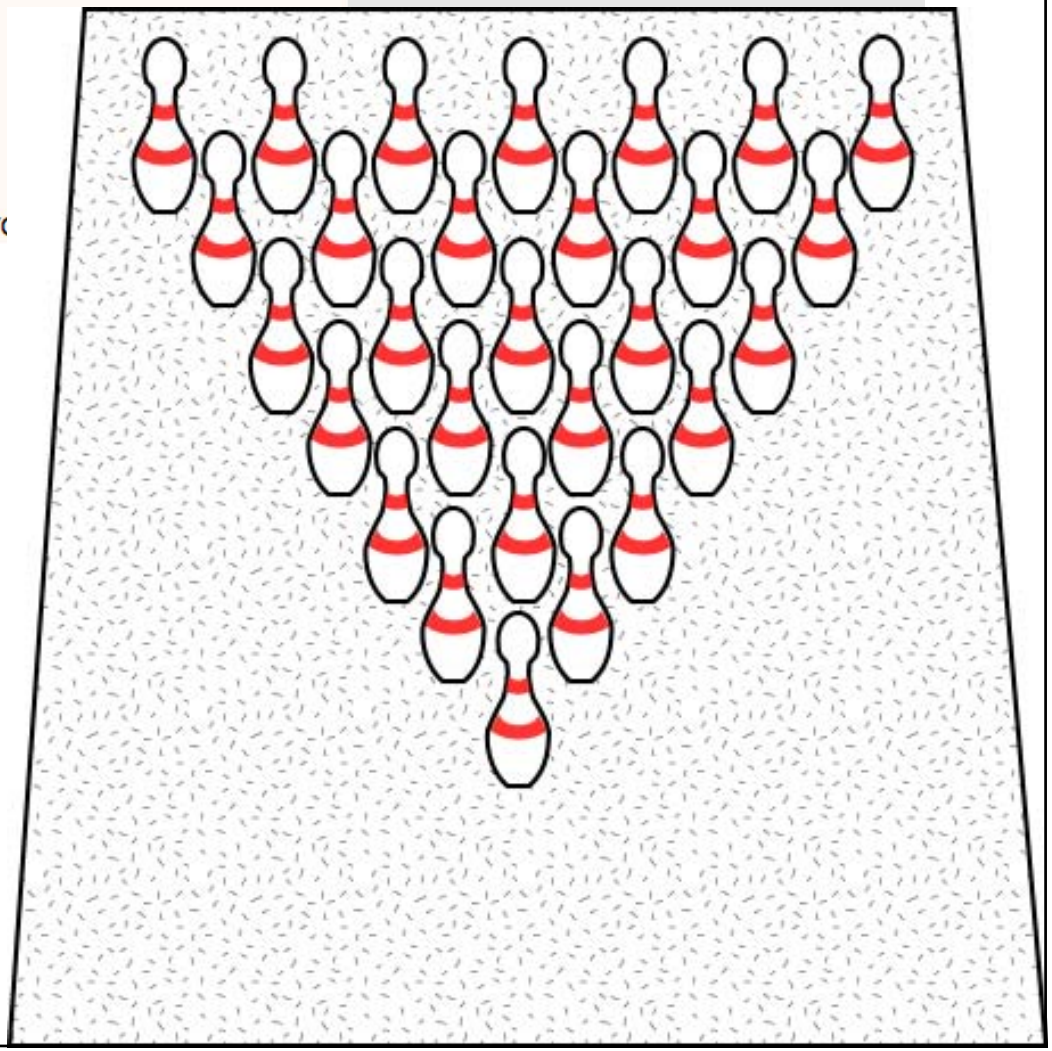
Click the image to see the added r



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

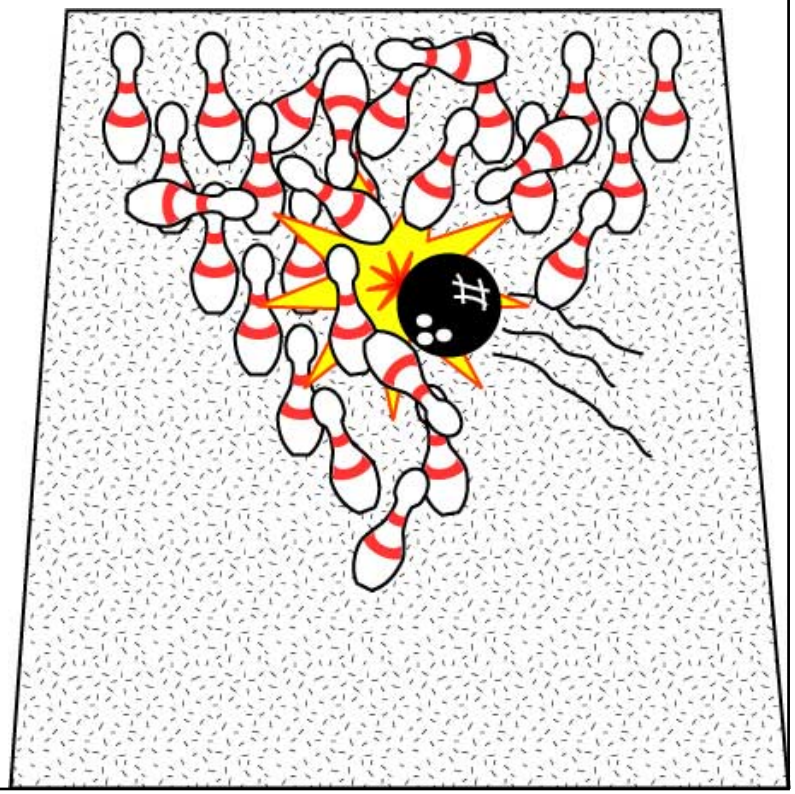
Click the image to see the added r



The arrangement of the pins for *Ten Pins* is at right. Say that another row of pins were added. How many pins are in that row?

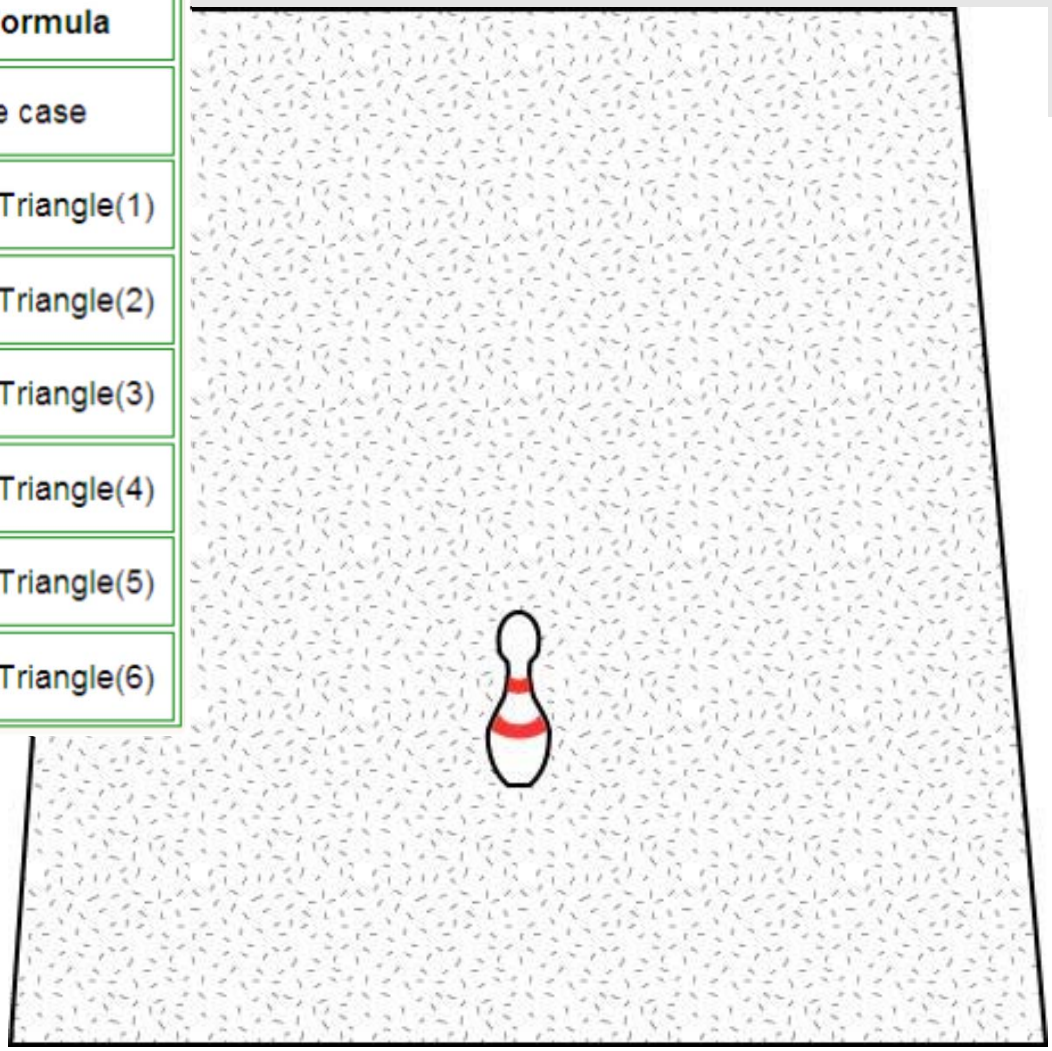
Row 1 has 1 pin.
Row 2 has 2 pins.
Row 3 has 3 pins.
Row 4 has 4 pins.
Row 5 has _____ pins.

Click the image to see the added row.



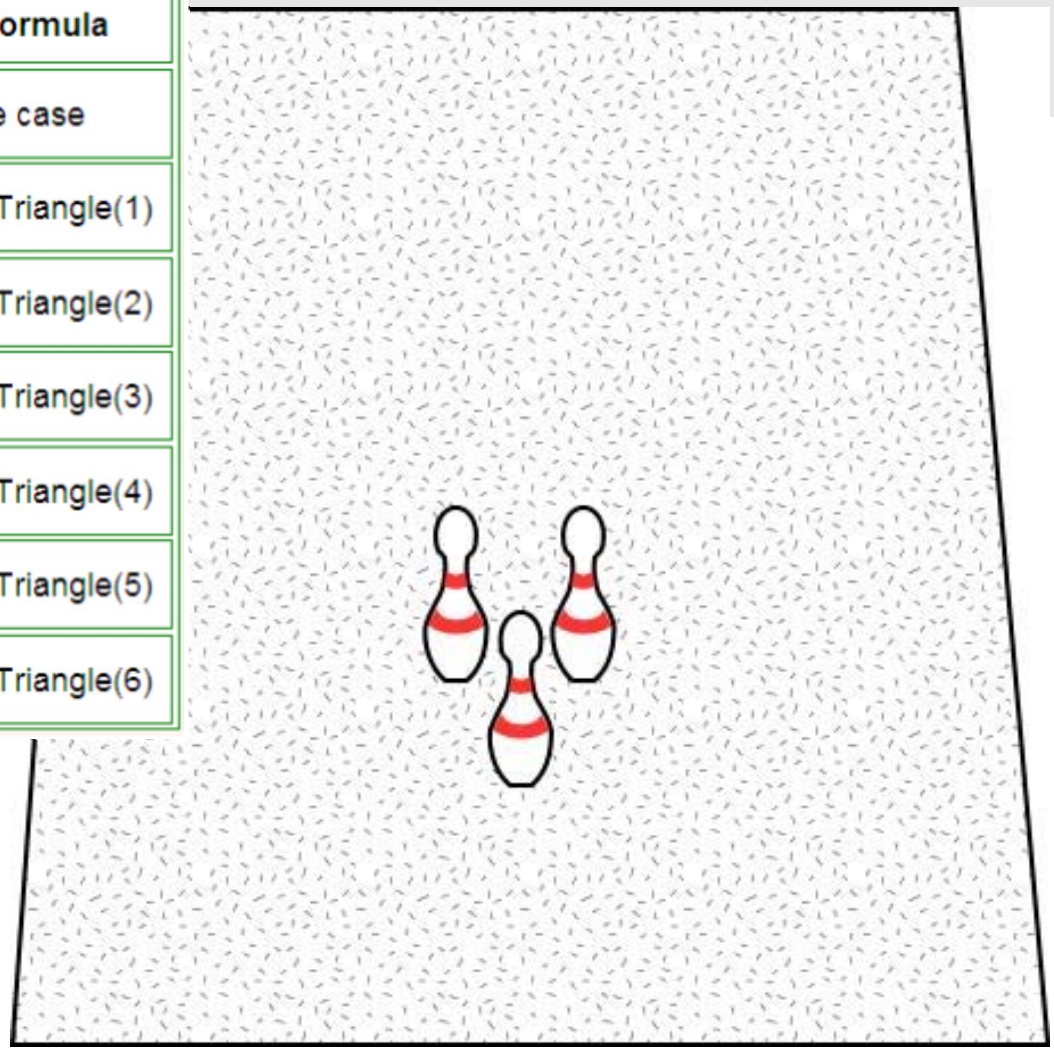
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



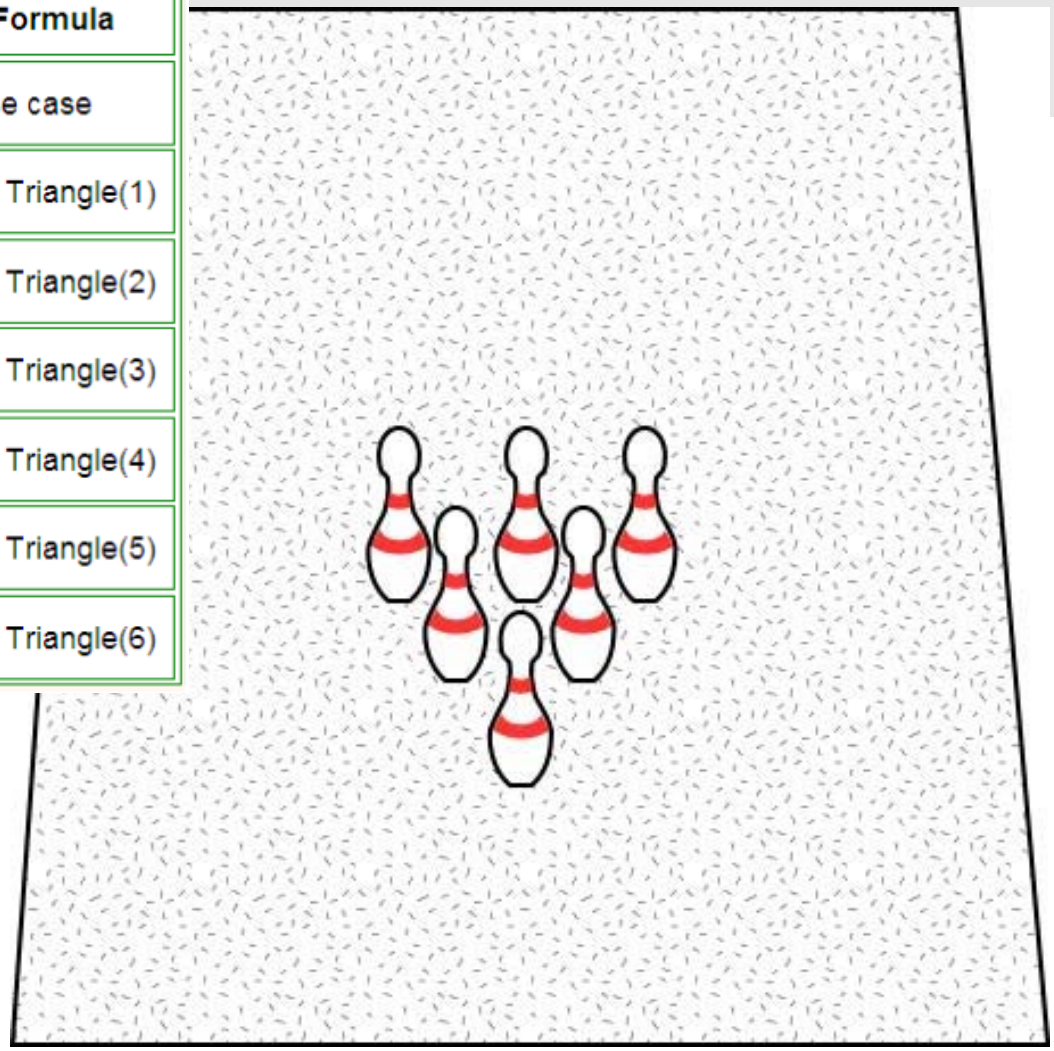
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



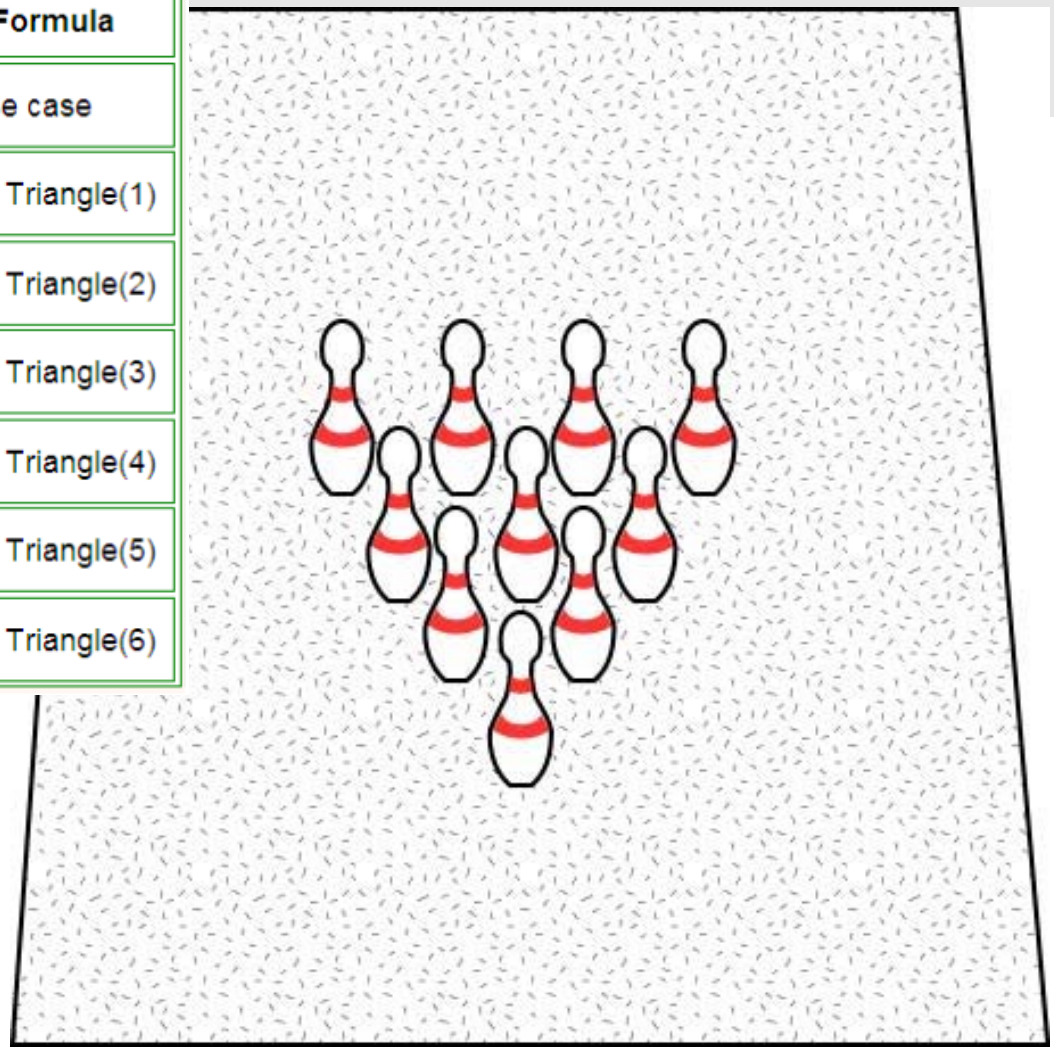
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



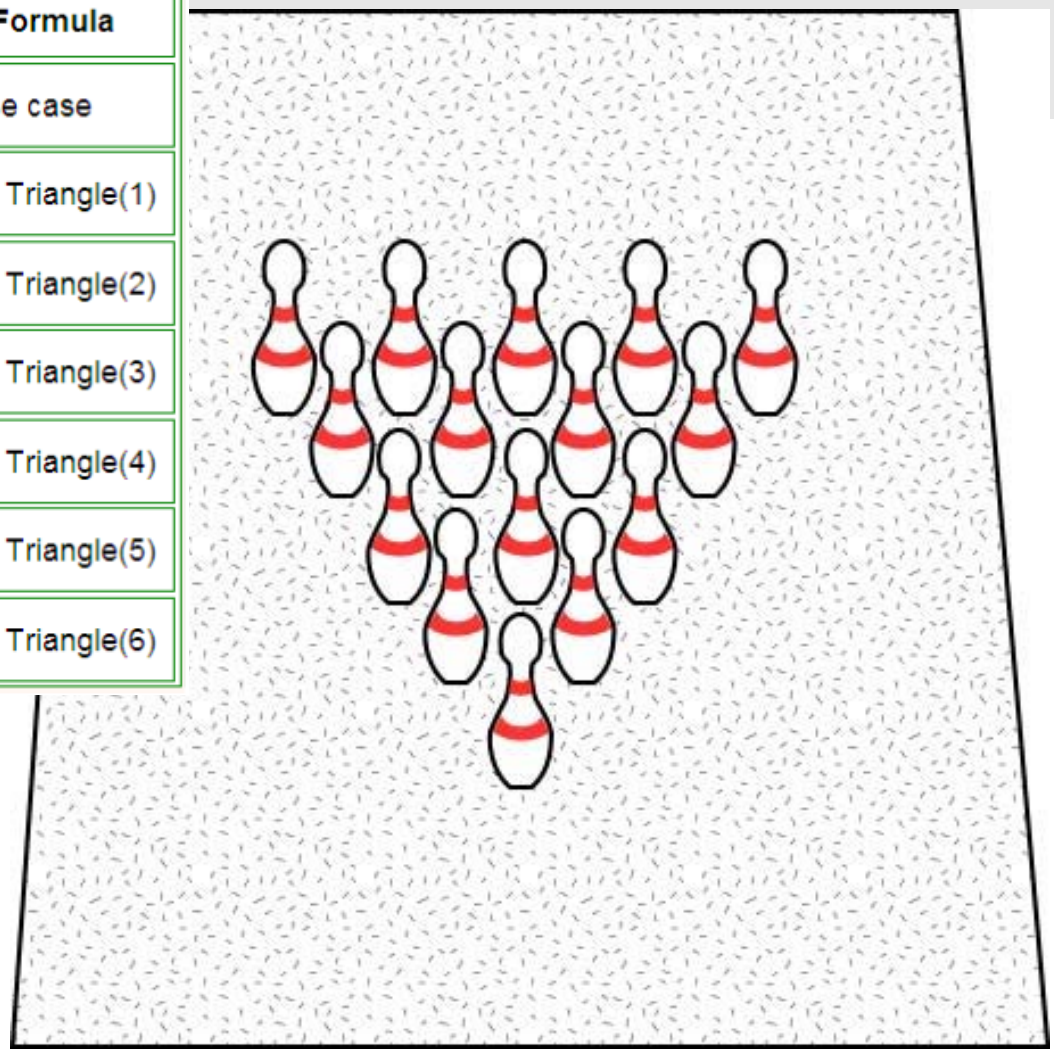
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



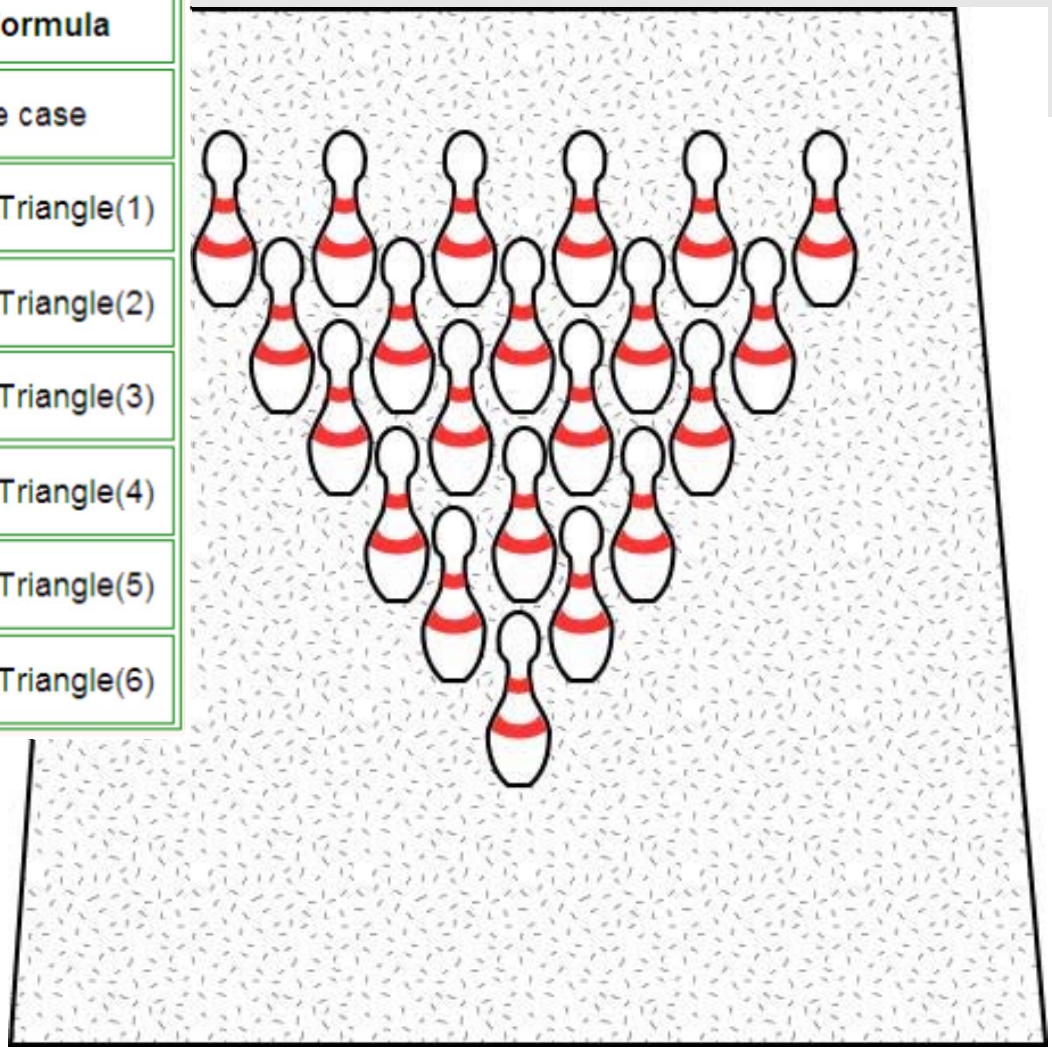
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



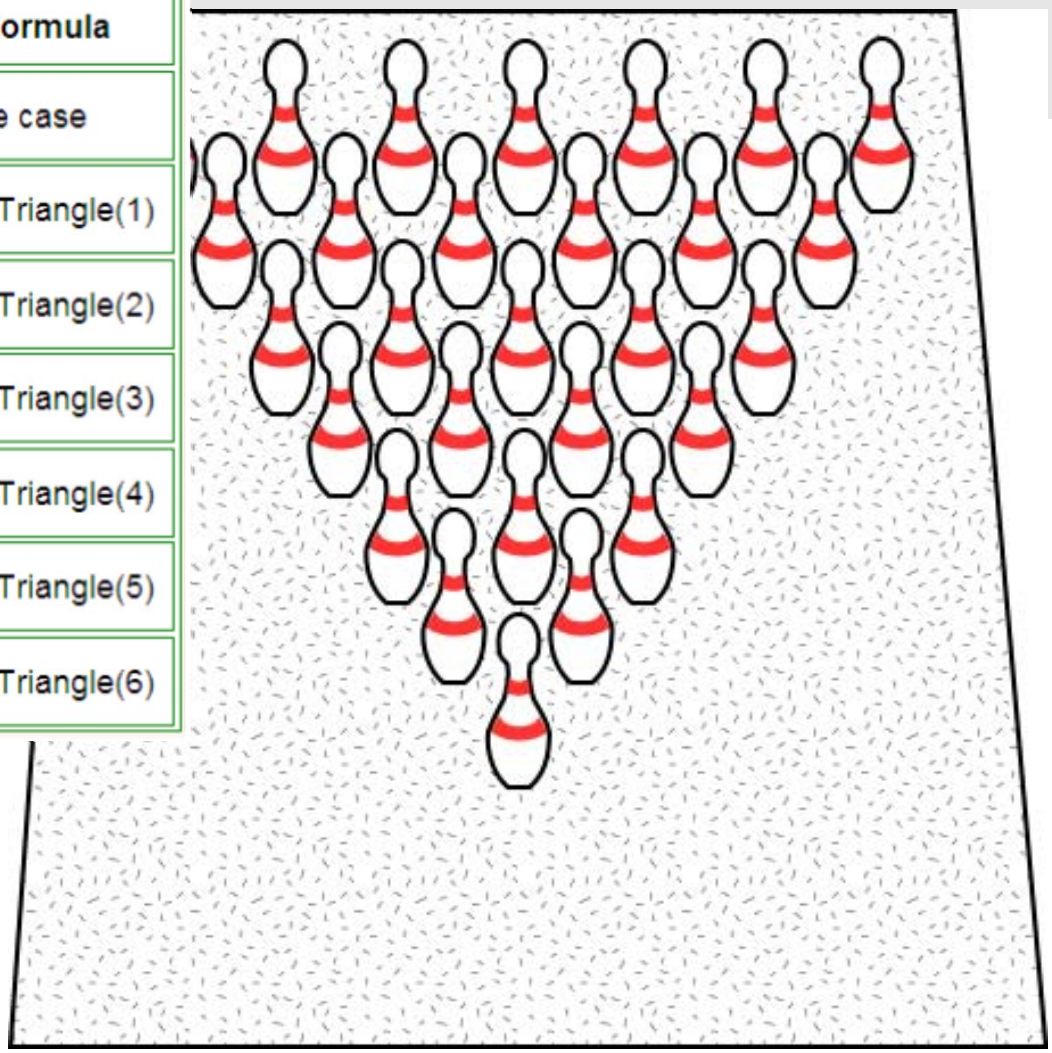
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



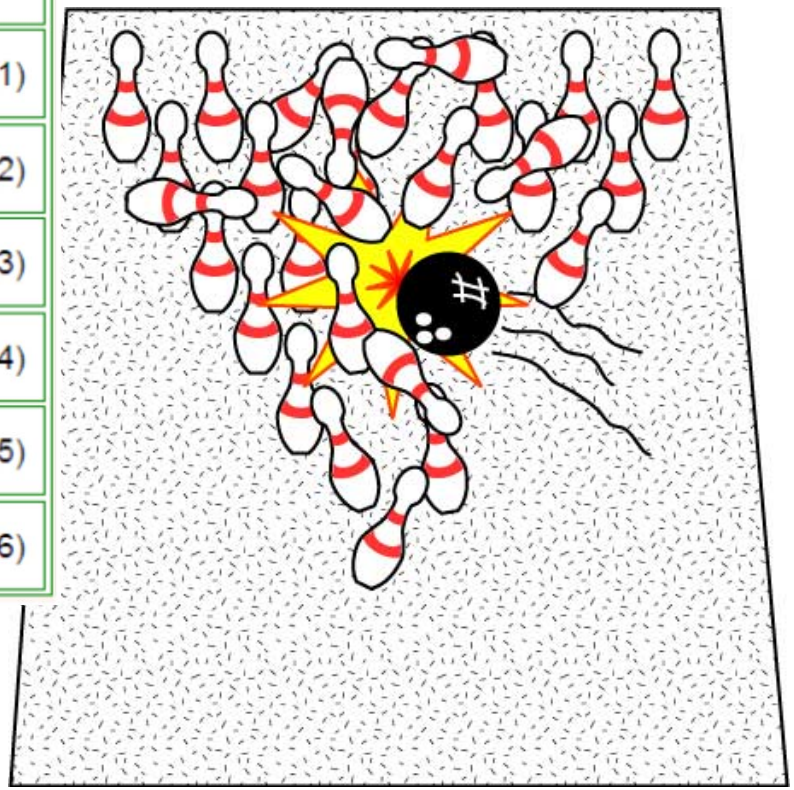
Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



Triangle Numbers

Rows	Formula
1	base case
2	2 + Triangle(1)
3	3 + Triangle(2)
4	4 + Triangle(3)
5	5 + Triangle(4)
6	6 + Triangle(5)
7	7 + Triangle(6)



Formula for Triangle Numbers

The table shows values of the `Triangle()` function. For example, `Triangle(4) = 10`. But it stops at seven. What is `Triangle(12)`?

We have figured out that the following is true:

```
total number of pins in N rows = number of pins in row N +
                                 total number of pins in N-1 rows

                               = N + total number of pins in N-1 rows
```

This can be written as a formula:

```
Triangle( N ) = N + Triangle( N - 1 )
```

So the number of pins in a 12 row arrangement is 12 + number of pins in 11 rows.

Base Case

This keeps on going. Each round divides the problem into an integer and a problem that is smaller by one. Eventually you reach the end:

```
number of pins in 12 rows = 12 + number of pins in 11 rows
number of pins in 11 rows = 11 + number of pins in 10 rows

                          . . .

number of pins in  3 rows =  3 + number of pins in 2 rows
number of pins in  2 rows =  2 + number of pins in 1 row
                          =  2 + 1
```

The *number of pins in 1 row* is called a **base case**. A *base case* is a problem that can solved immediately. In this example, the number of pins in 1 row is 1.

Recursion with Triangle Numbers

Here are the two parts to recursion:

1. If the problem is easy, solve it immediately.
    - An easy problem is a **base case**.
2. If the problem can't be solved immediately, divide it into smaller problems, then:
    - Solve the smaller problems by applying this procedure to each of them.

And here is how this applies to triangle numbers:

1. Triangle(1) = 1
2. Triangle(N) = N + Triangle(N-1)

The problem "`Triangle(N)`" is divided into two problems: "add N to something" and "`Triangle(N-1)`". Sometimes the latter can be solved immediately (when it is the base case). Other times you need to re-apply the solution to the smaller problem.

Example

Here is our solution for triangle numbers:

1. Triangle(1) = 1
2. Triangle(N) = N + Triangle(N-1)

Here is another example, calculating `Triangle( 5 )`:

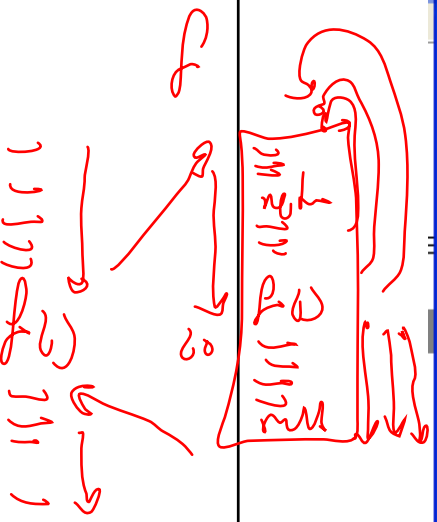
```
Triangle( 5 ) = 5 +  Triangle( 4 )
              = 5 + ( 4 + Triangle( 3 ) )
              = 5 + ( 4 + ( 3 + Triangle( 2 ) ))
              = 5 + ( 4 + ( 3 + ( 2 + Triangle( 1 ) )))
              = 5 + ( 4 + ( 3 + ( 2 +      1        )))

              = 5 + ( 4 + ( 3 + 3) )
              = 5 + ( 4 + 6 )
              = 5 + 10
              = 15
```

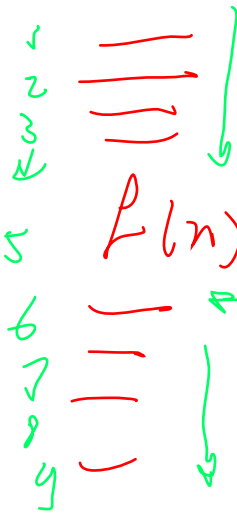
Python Method

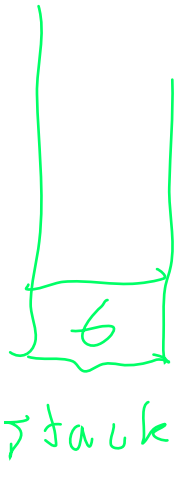
*simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\simple.py

File Edit Format Run Options Window Help

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)

n=int(input('Enter a number: '))
result=Triangle(n)
print('Triangle result is:', result)

#output

    Enter a number: 5
    Triangle result is: 15
```





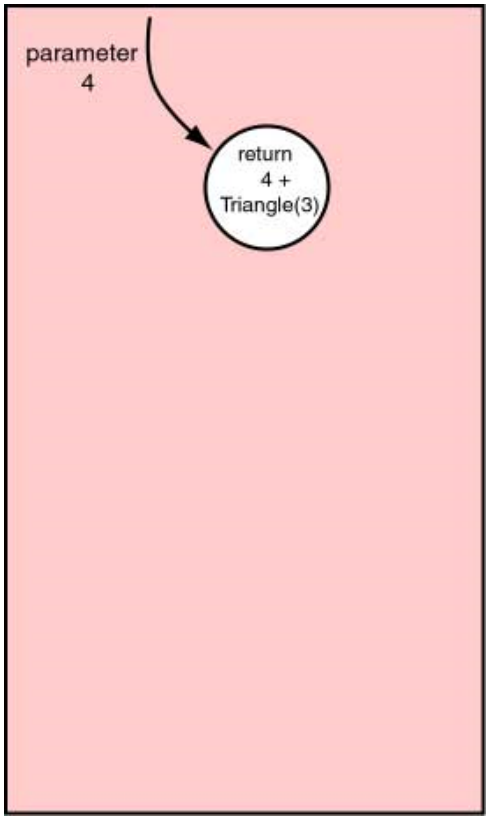


Static view of the Method

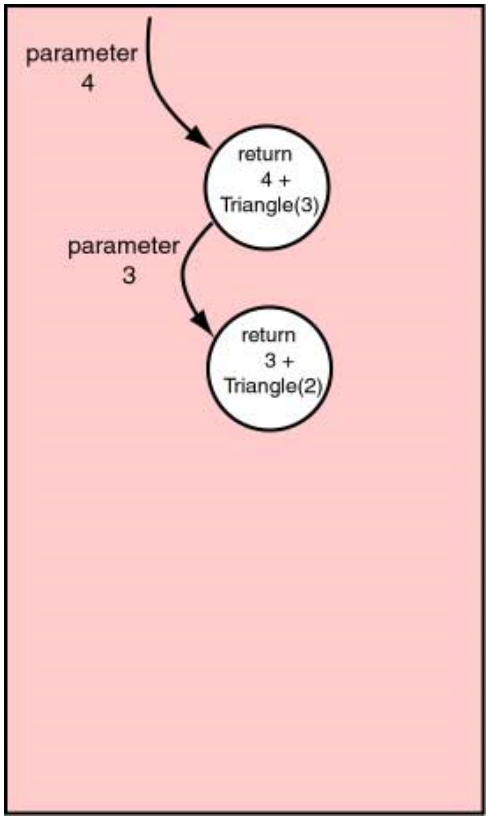
```
Triangle( 1 ) = 1
Triangle( N ) = N + Triangle( N-1 )
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```

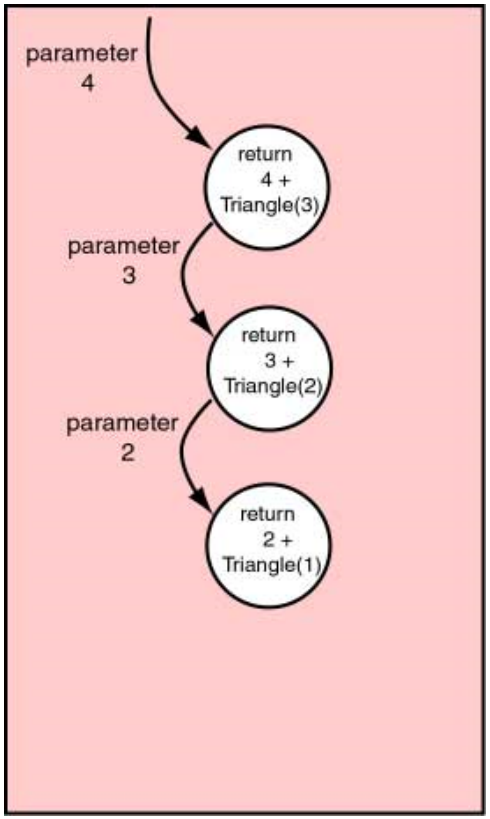
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



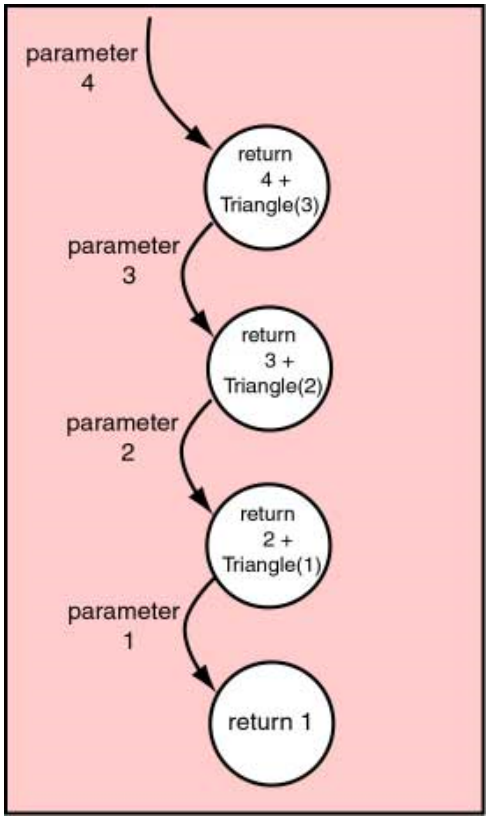
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



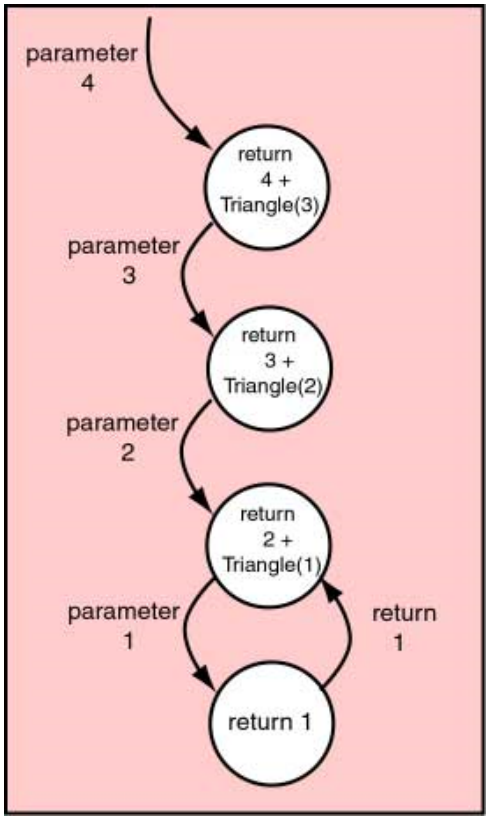
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



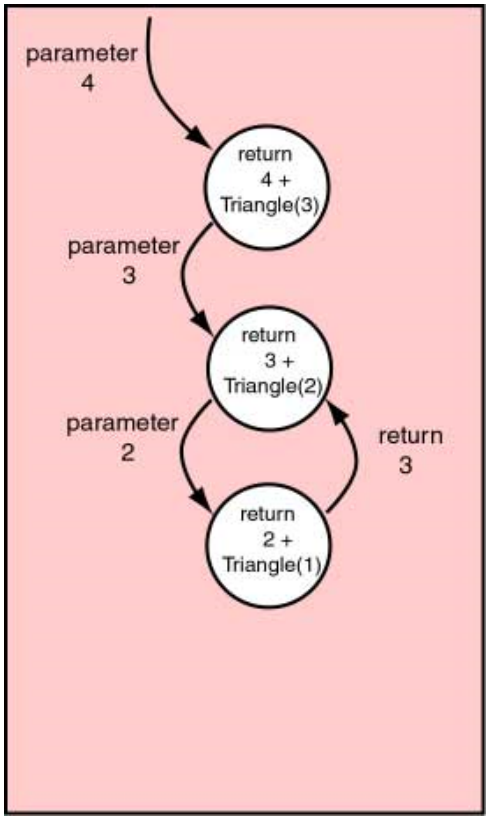
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



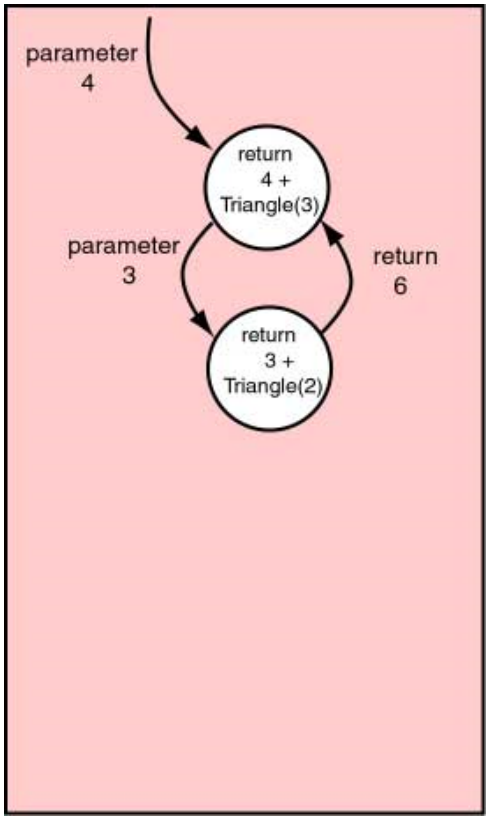
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



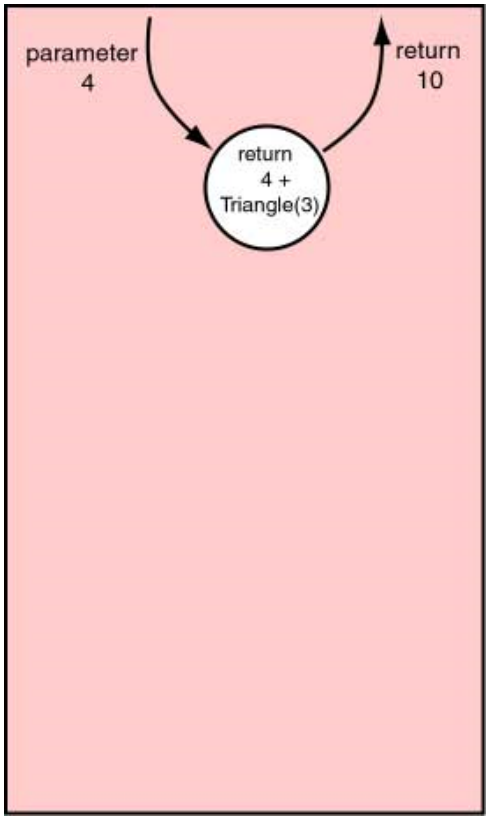
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



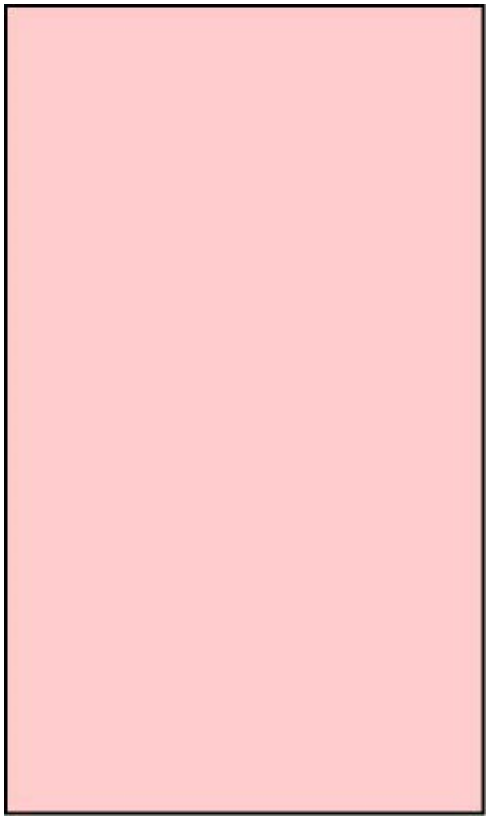
```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



Iterative Triangle

Here is an iterative method that calculates Triangle

```
def Triangle(N):
    totalPins=0
    for row in range(1, N+1):
        totalPins+=row
    return totalPins
```

You might prefer it to the recursive version because the code looks familiar. But notice that it is not as clear that it implements the definition.

Formula for Triangle

`Triangle` is so easy that you don't need to use either iteration or recursion for it. You might recognize `Triangle(N)` as the sum of the series

```
1 + 2 + 3 + ... + (N-1) + N = (N(N+1)) / 2
```

For example

```
1 + 2 + 3 + 4  = (4(5)) / 2 = 10
```

Factorial Function

You have probably seen the factorial function before. It is defined for all integers greater or equal to zero:

```
factorial( 0 ) = 1
factorial( N ) = N * factorial( N-1 )
```

For example,

```
factorial( 5 ) = 5 * factorial( 4 )
               = 5 * ( 4 * factorial( 3 ) )
               = 5 * ( 4 * (3 * factorial( 2 ) ) )
               = 5 * ( 4 * (3 * (2 * factorial( 1 ) ) ) )
               = 5 * ( 4 * (3 * (2 * ( 1 * factorial( 0 ) ) ) ) )
               = 5 * ( 4 * (3 * (2 * ( 1 * 1 ) ) ) )
               = 5 * 4 * 3 * 2 * 1 * 1
               = 120
```

Often factorial(N) is written N!

Examine the definition and check that it includes both of the two parts of a recursive definition.

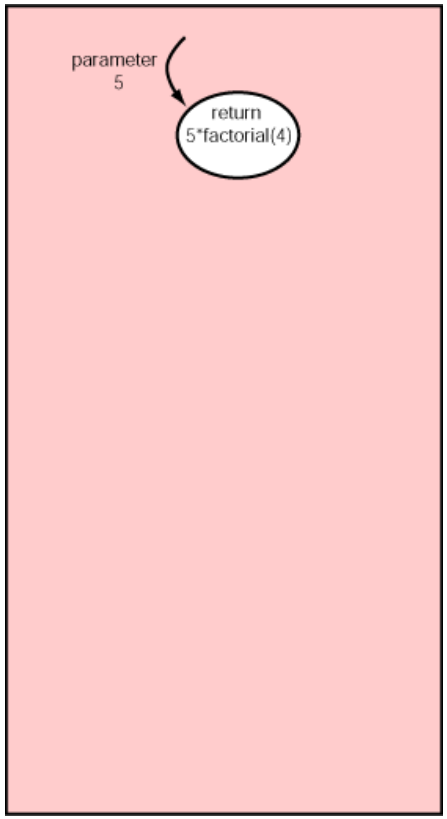
Static View of Recursion

In the "static view" of recursion, you translate a math-like definition into the Java method definition. You don't think much about what happens when the method runs. Here is the math-like definition of recursion:

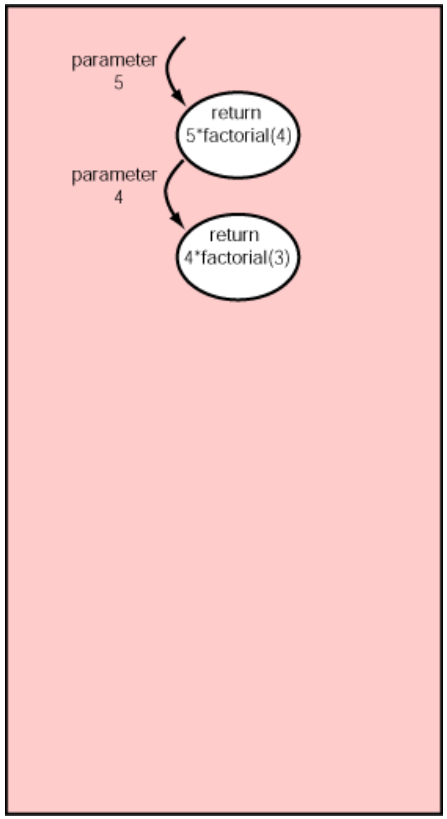
```
factorial( 0 ) = 1
factorial( N ) = N * factorial( N-1 )
```

```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```

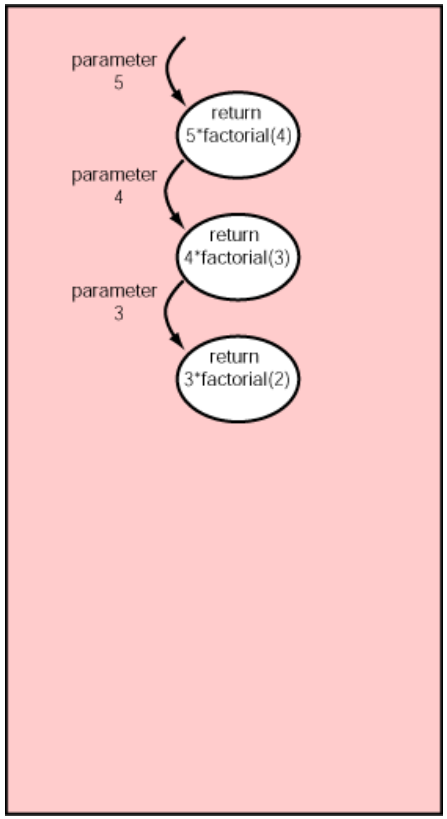
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



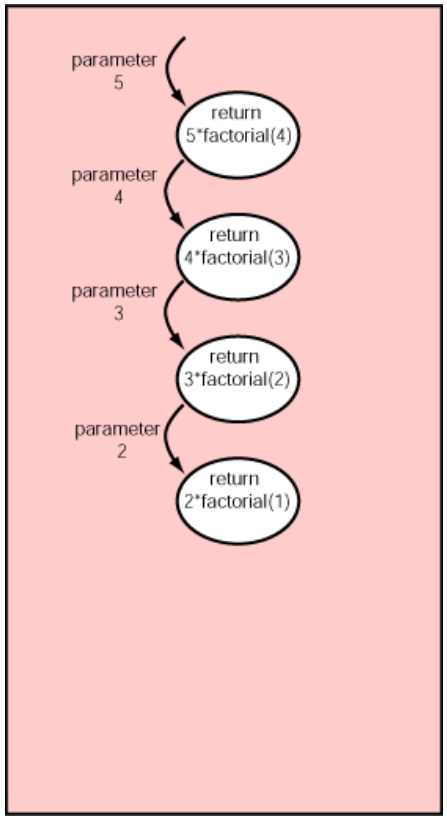
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



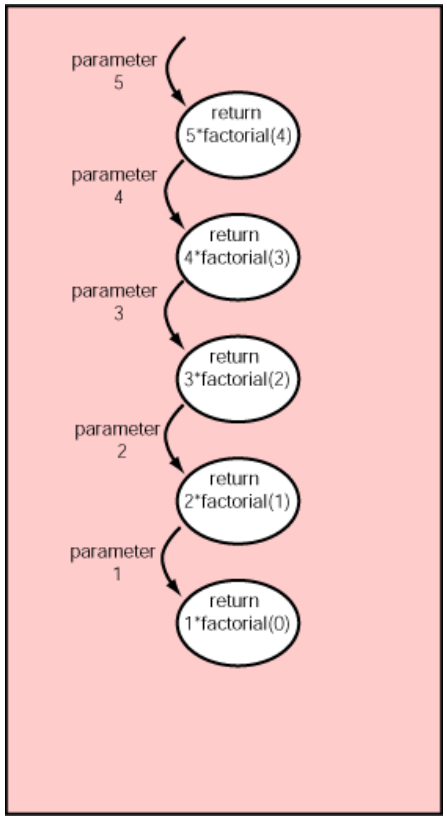
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



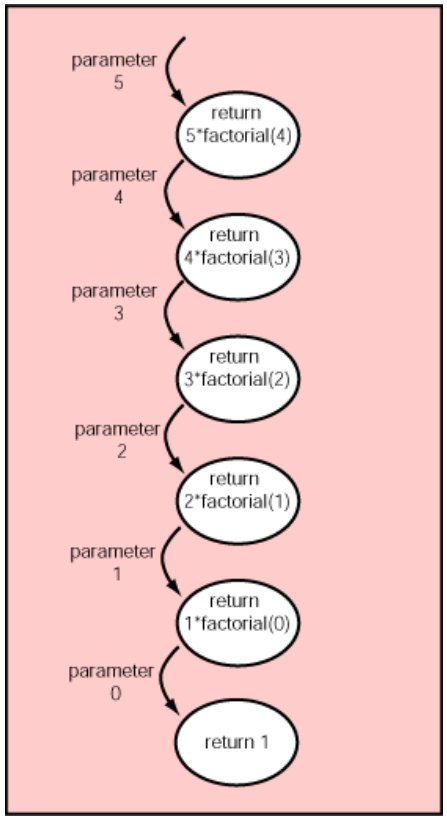
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



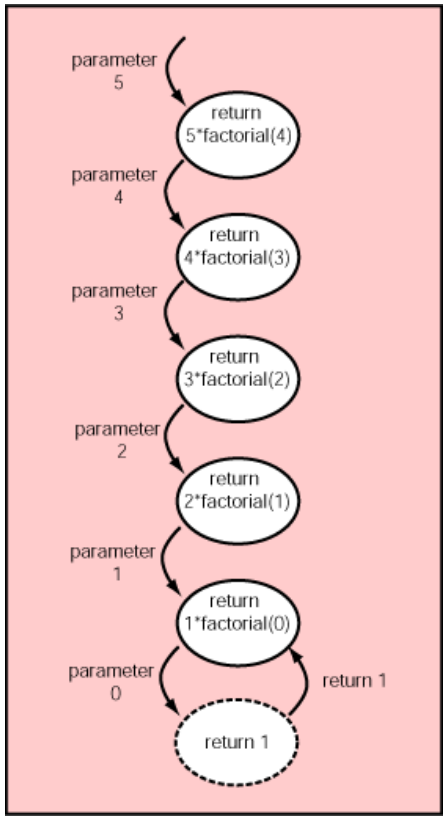
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



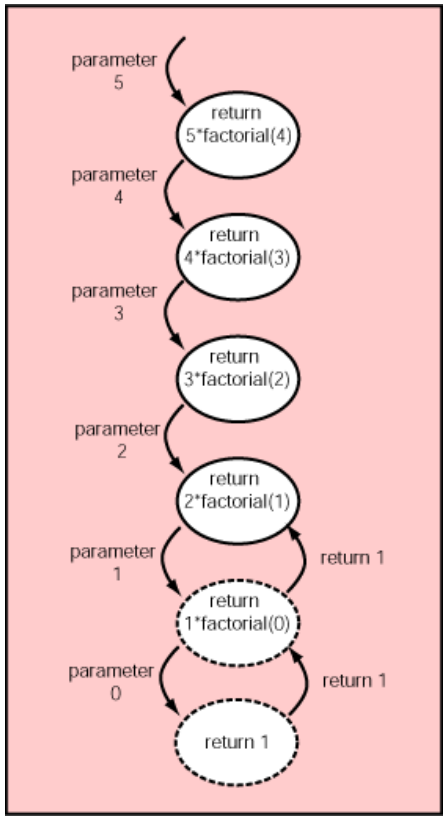
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



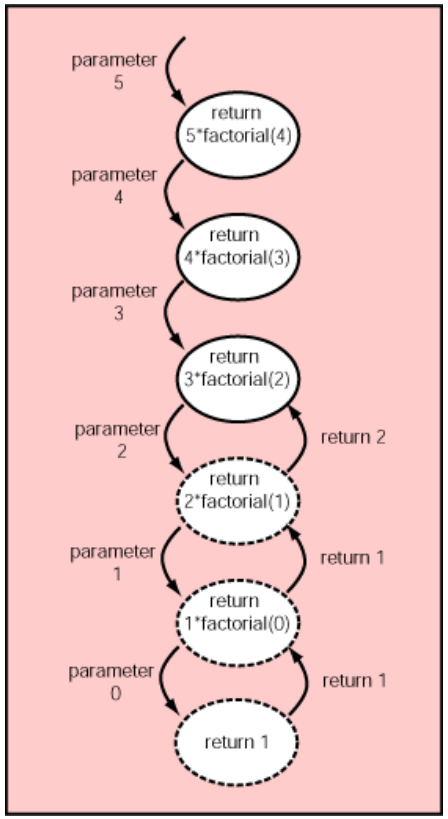
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



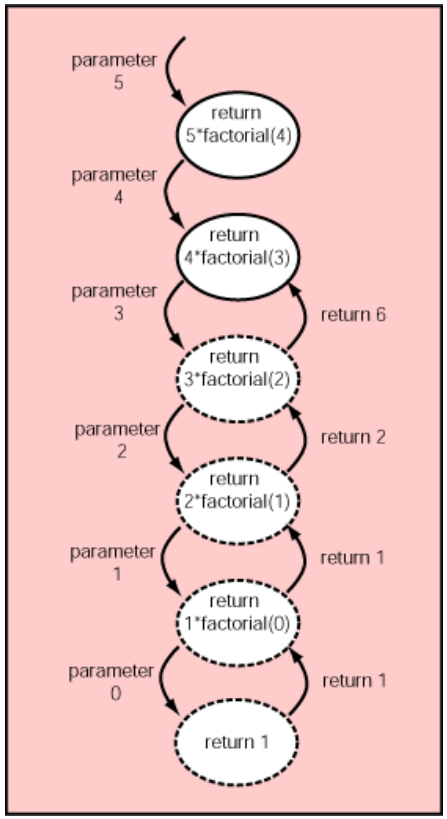
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



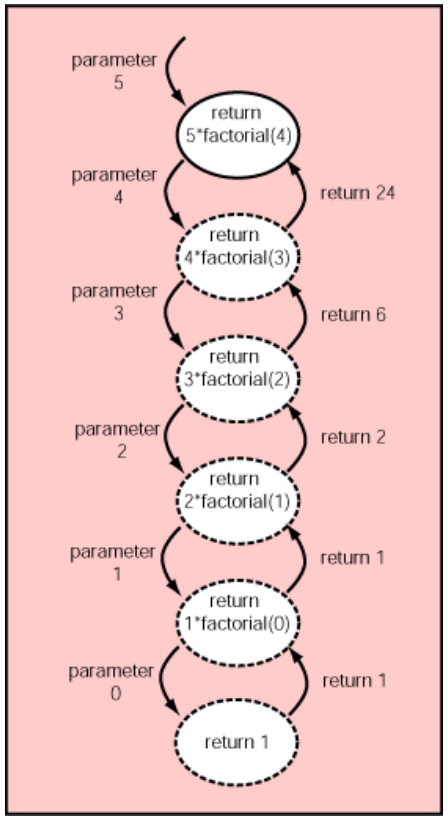
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



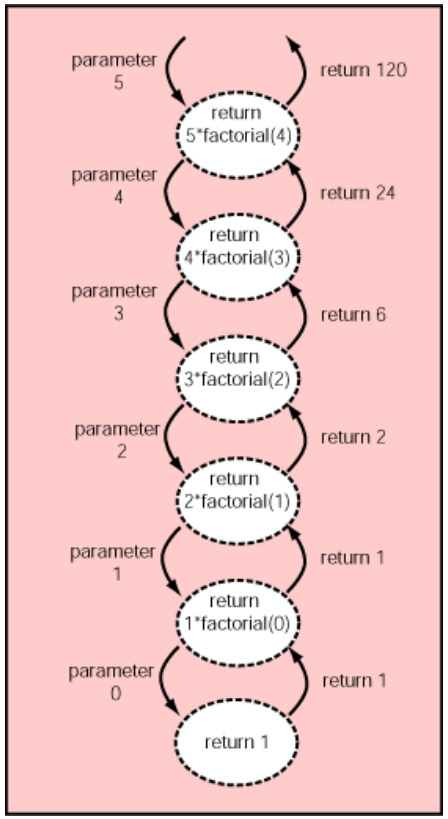
```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



```
def factorial(N):
    if (N==0):
        return 1
    else:
        return N * factorial(N-1)
```



Defensive Programming

Defensive programming is when the programmer anticipates problems and writes code to deal with them. To avoid the disaster a negative parameter would cause, sometimes `factorial()` is written like this:

```
def factorial(N):
    if (N<=0):
        return 1
    else:
        return N * factorial(N-1)
```

But, according to the math-like definition, this is not correct. The value of `factorial( -1 )` is undefined, not equal to one. Potentially, returning an incorrect value and continuing on as if nothing is wrong might cause a greater disaster than stopping the computer system.

Sometimes the method is written to throw an exception when an illegal argument is detected. But this adds complication since now the caller must deal with a possible exception.

Perhaps the best solution is to write `factorial()` so that it follows the math-like definition exactly. It is the responsibility of the caller to make sure that the argument is within range.

Complete Program

```
def factorial(N):
    if (N<=0):
        return 1
    else:
        return N * factorial(N-1)

n=int(input('Enter a number: '))
result=factorial(n)
print('Factrorial(', n, ') is: ', result)

#output
Enter a number: 100
Factrorial( 100 ) is: 93326215443944152681699238856266
                      70049071596826438162146859296389
                      52175999932299156089414639761565
                      18286253697920827223758251185210
                      916864000000000000000000000000
```

Iterative Implementation of Factorial

Here is the math-like definition of recursion (again):

```
factorial( 0 ) = 1
factorial( N ) = N * factorial( N-1 )
```

And here is an iterative implementation:

```
def factorial(N):
    p=1
    for i in range(1, N+1):
        p *=i
    return p
}
```

Of course, this version suffers from the same overflow problem as the recursive version. For any argument larger than 12 the product is too large to hold in an **int** and incorrect values will be returned to the caller.

simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\simple.py (3.4.4)

File Edit Format Run Options Window Help

```
def factorial(N):
    p=1
    for i in range(1, N+1):
        p *=i
    return p

n=int(input('Enter a number: '))
result=factorial(n)
print('Factrorial(', n, ') is: ', result)

#output

Enter a number: 300
Factrorial( 300 ) is:  306057512216440636035370461297268629388588804173
                       576999416776741259476533176716867465515291422477
                       573349939147888701726368864263907759003154226842
                       927906974559841225476930271954604008012215776252
                       176854255965356903506788725264321896264299365204
                       576448830388909753943489625436053225980776521270
                       822437639449120128678675368305712293681943649956
                       460498166450227716500185176546469340112226034729
                       724066333258583506870150169794168850353752137554
                       910289126407157154830282284937952636580145235233
                       156936482233436799254594095276820608062232812387
                       383880817049600000000000000000000000000000000000
                       000000000000000000000000000000000000
                       0000
```

```
Enter a number: 1000
Factrorial( 1000 ) is:  40238726007709377354370243392300398571937486421071463254
37999104299385123986290205920442084869694048004799886101971960586316668729948085
58901323829669944590997424504087073759918823627727188732519779505950995276120874
97546249704360141827809464649629105639388743788648733711918104582578364784997701
24766328898359557354325131853239584630755574091142624174743493475534286465766116
67797396668820291207379143853719588249808126867838374559731746136085379534524221
58659320192809087829730843139284440328123155861103697680135730421616874760967587
13483120254785932076716913244842623613141250878020800026168315102734187977704 7
84635868170164365024153691398281264810213092761244896359928705114964975419909342
22156683257208082133318611681155361583654698404670897560290095053761647584772842
18896796462449451607653534081989013854424879849599533191017233555566021394503997
36280750137837615307127761926849034352625200015888535147331611702103968175921510
90778801939317811419454525722386554146106289218796022383897147608850627686296714
66746975629112340824392081601537808898939645182632436716167621791689097799119037
54031274622289988005195444414282012187361745992642956581746628302955570299024324
15318161721046583203678690611726015878352075151628422554026517048330422614397428
69330616908979684825901254583271682264580665267699586526822728070757813918581788
89652208164348344825993266043367660176999612831860788386150279465955131156552036
09398818061213855860030143569452722420634463179746059468257310379008402443243846
56572450144028218852524709351906209290231364932734975655139587205596542287497740
11413346962715422845862377387538230483865688976461927383814900140767310446640259
89949022222176590433990188601856652648506179970235619389701786004081188972991831
10211712298459016419210688843871218556461249607987229085192968193723886426148396
57382291123125024186649353143970137428531926649875337218940694281434118520158014
12334482801505139969429015348307764456909907315243327828826986460278986432113908
35062170950025973898635542771967428222487575867657523442202075736305694988250879
68928162753848863396909959826280956121450994871701244516461260379029309120889086
94202851064018215439945715680594187274899809425474217358240106367740459574178516
08292301353580818400969963725242305608559037006242712434169090041536901059339838
35777939410970027753472000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000
```

Rabbits

Say that a female rabbit is mature 2 months after birth. Assume that each mature female rabbit produces 1 baby female rabbit per month. If you start out with one baby female rabbit, how many female rabbits will you have in 10 months? Assume that there are enough male rabbits to ensure maximum production, and that no rabbits die.

Here is a chart:

N	1	2	3	4	5	6	7	8	9	10
Females	1									

The chart shows the number of rabbits at the end of the first month, at the end of the second month, and so on.

End of Month Two

Recall the rules:

- A female rabbit matures after 2 months.
- Each mature female rabbit produces 1 baby female rabbit per month.

The chart shows the situation at the end of the second month. Your single female rabbit has just become mature. Assume that this fact has not gone unnoticed by the male rabbits.

N	1	2	3	4	5	6	7	8	9	10
Females	1	1								

End of the Third Month

The new chart shows this. At the end of month three there is 1 mature female rabbit and 1 baby female rabbit for a total of two female rabbits.

N	1	2	3	4	5	6	7	8	9	10
Females	1	1	2							

If you look under month 2 you see how many mature female rabbits there are in month 4. Since it takes rabbits two months to become mature, every rabbit listed for month N will be mature in month N+2. Each one of these rabbits will produce one new baby rabbit in month N+2.

If you look under month 3 you see how many total female rabbits there are at the start of month 4.

Add the rabbits in month 2 (which equals the number of babies born during month 4) to the rabbits in month 3 to get the total number of rabbits ate the end of month 4.

End of Month Four

Month 4 starts out with two rabbits (the number under month 3) one of which is mature (the number under month 2). During month 4 the mature rabbit gives birth to another baby. The chart shows the total of three rabbits at the end of month 4.

N	1	2	3	4	5	6	7	8	9	10
Females	1	1	2	3						
		*	**							

```
*   –  – Mature females at start of month 4
**  –  – Total females at start of month 4
```

The total number of female rabbits at the start of a month is shown under the previous month in the chart. The number of mature rabbits (which give birth during that month) is the number shown under the second previous month.

Month 5

Month 5 starts out with three rabbits (the number under month 4) two of which are mature (the number under month 3). During the month the mature rabbits give birth to one baby each. The chart shows the total of five rabbits at the end of the month.

N	1	2	3	4	5	6	7	8	9	10
Females	1	1	2	3	5					
			**	***						

```
**   – Mature females at start of month 5
***  – Total females at start of month 5
```

At the end of month 5 there are five female rabbits.

Rabbits Rule

The updated chart shows the situation.

N	1	2	3	4	5	6	7	8	9	10
Rabbits	1	1	2	3	5	**8**				

Here is what we have been doing:

```
rabbits At End Of Month( N ) = rabbits At End Of Month( N-1 )
                                          +
                               rabbits Born During Month( N )

rabbits At End Of Month( N ) = rabbits At End Of Month( N-1 )
                                          +
                               mature Rabbits During Month( N )

rabbits At End Of Month( N ) = rabbits At End Of Month( N-1 )
                                          +
                               rabbits At End Of Month( N-2 )
```

Once you have this rule, to calculate the number of rabbits at the end of a month just add up the numbers for the previous two months.

Fibonacci

This problem was studied and solved by a famous mathematician, Fibonacci. The numbers of rabbits form the **Fibonacci series**. This series has important applications in several problems that have nothing to do with rabbits. Call the N'th number in the series `fib(N)`:

Fibonacci Series										
N	1	2	3	4	5	6	7	8	9	10
fib(N)	1	1	2	3	5	8	13	21	34	55

Here is the rule for filling the chart, rewritten for `fib(N)`:

```
fib( N ) = fib( N-1 ) + fib( N-2 )
```

Look at the chart to verify that the rule works.

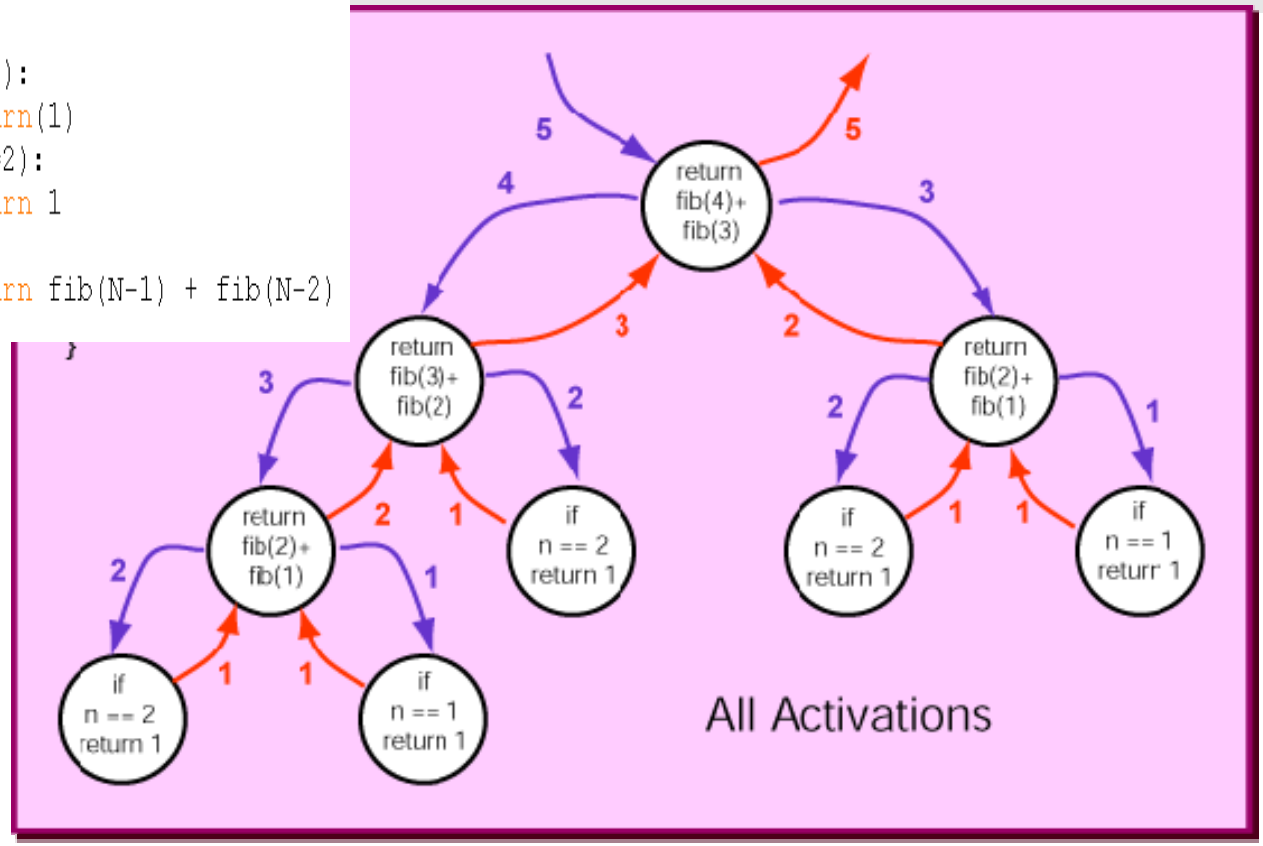
Complete Program

*fibbR.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\fibbR

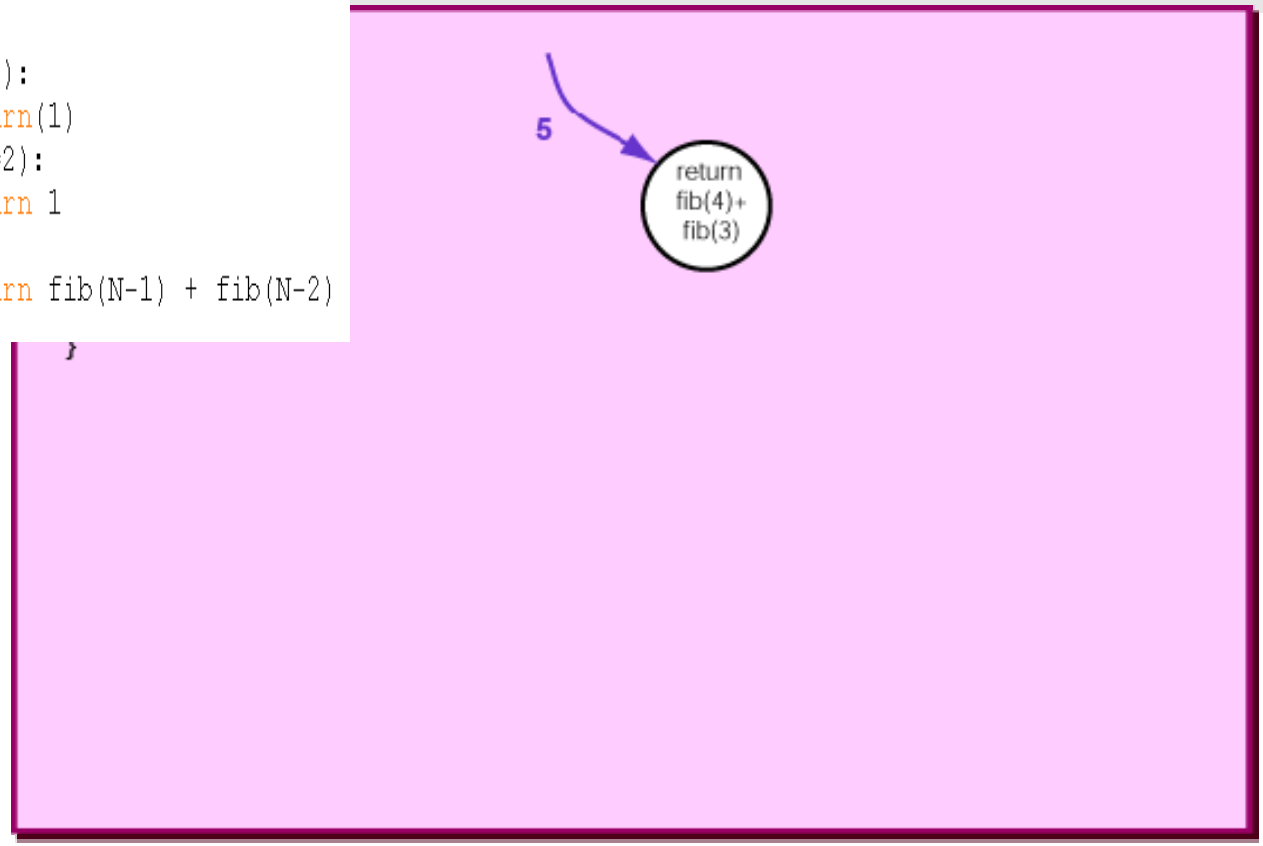
File Edit Format Run Options Window Help

```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)

n=int(input('Enter a number: '))
result=fib(n)
print('Fib(', n, ') is: ', result)

#output

    Enter a number: 9
    Fib( 9 ) is:  34
```

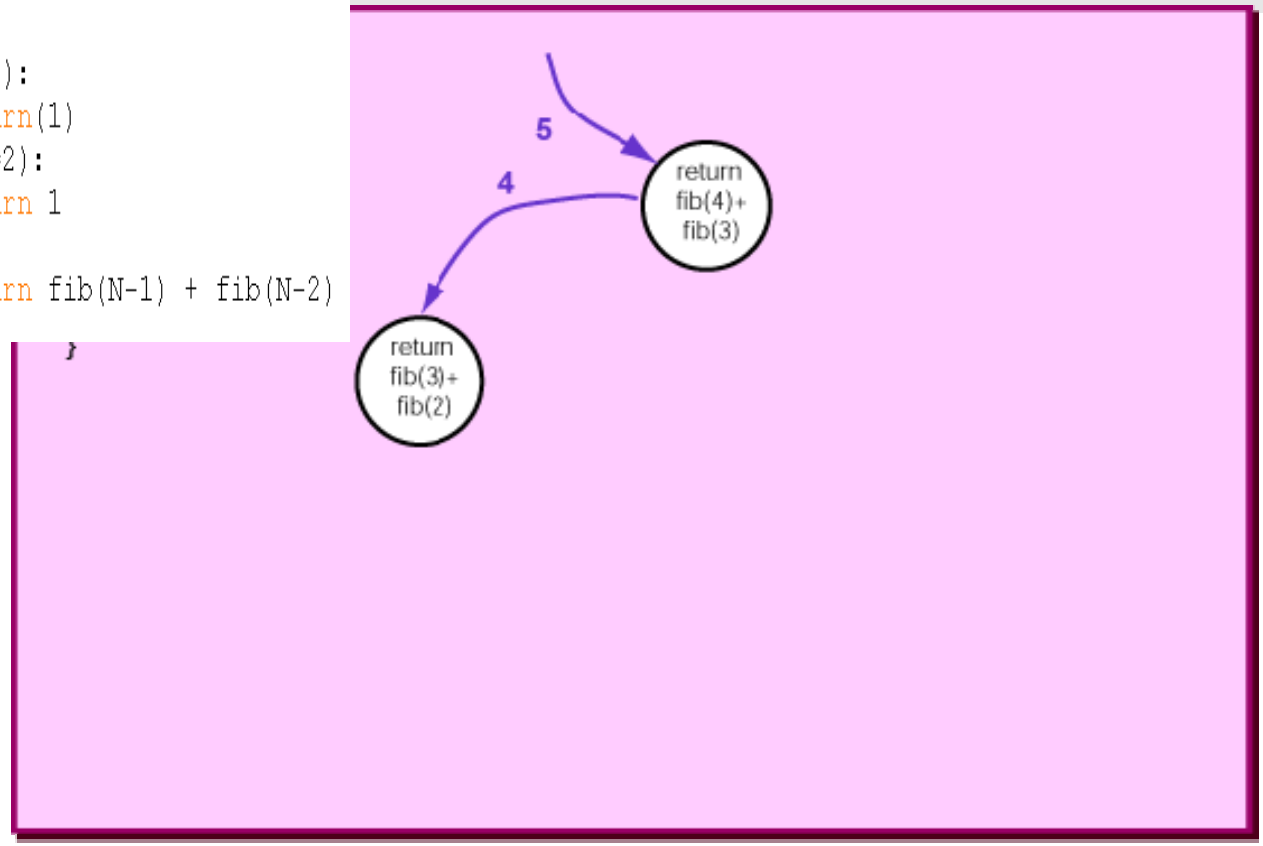
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



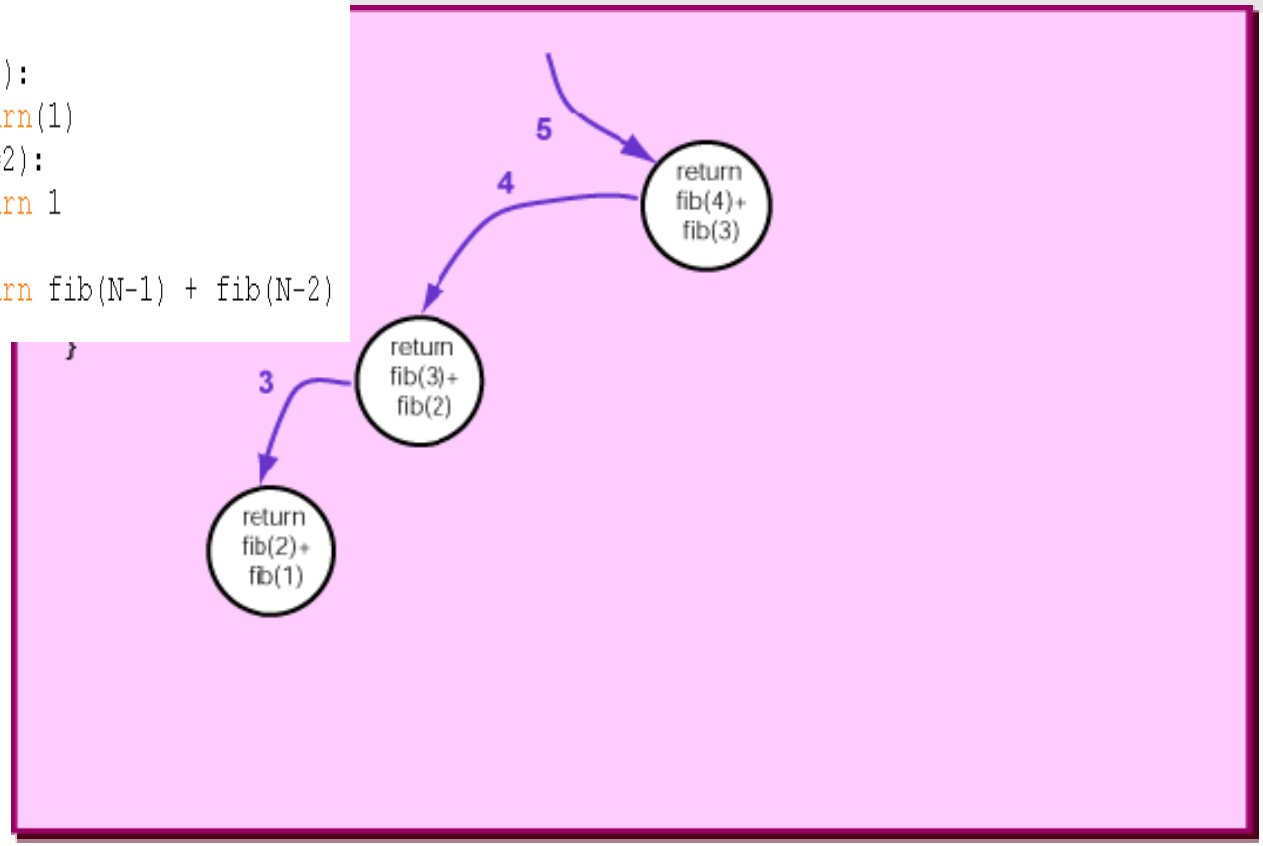
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



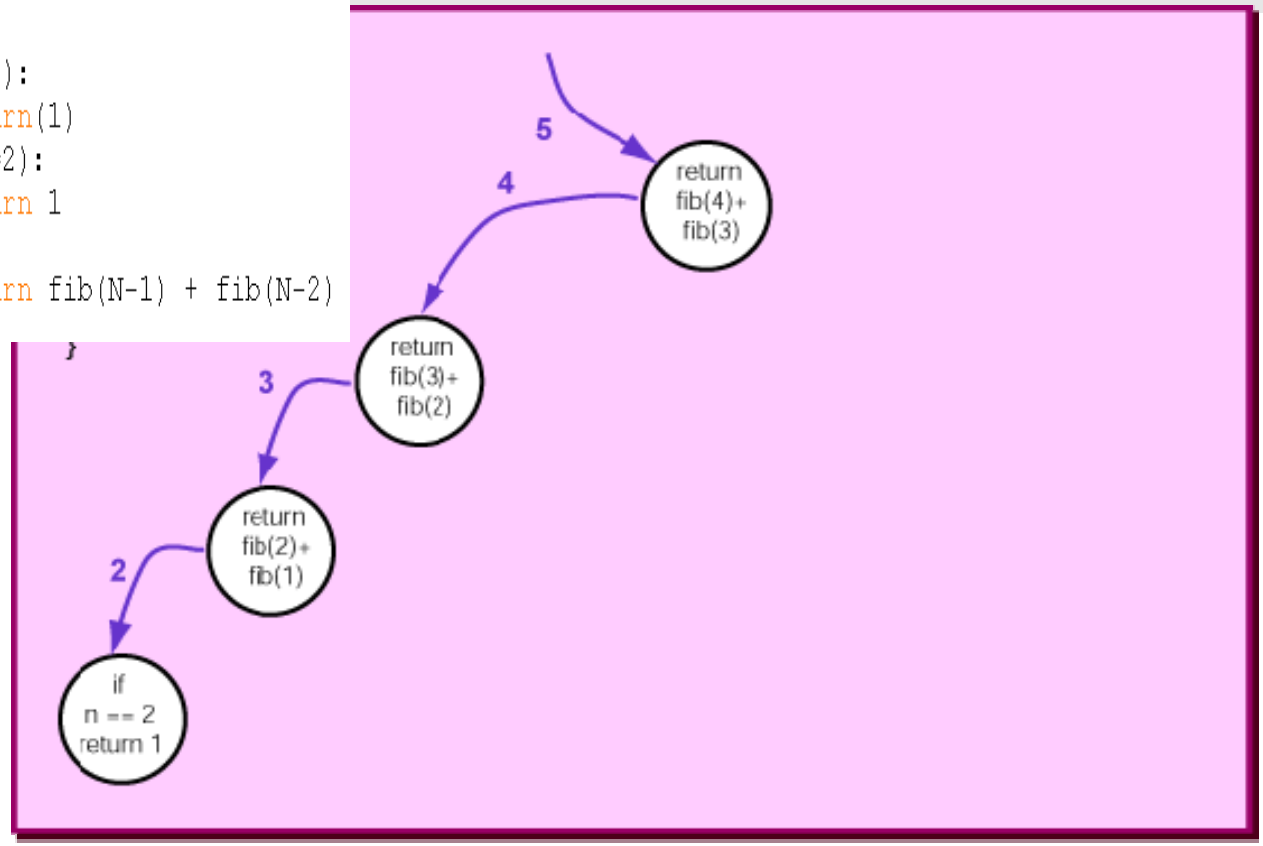
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



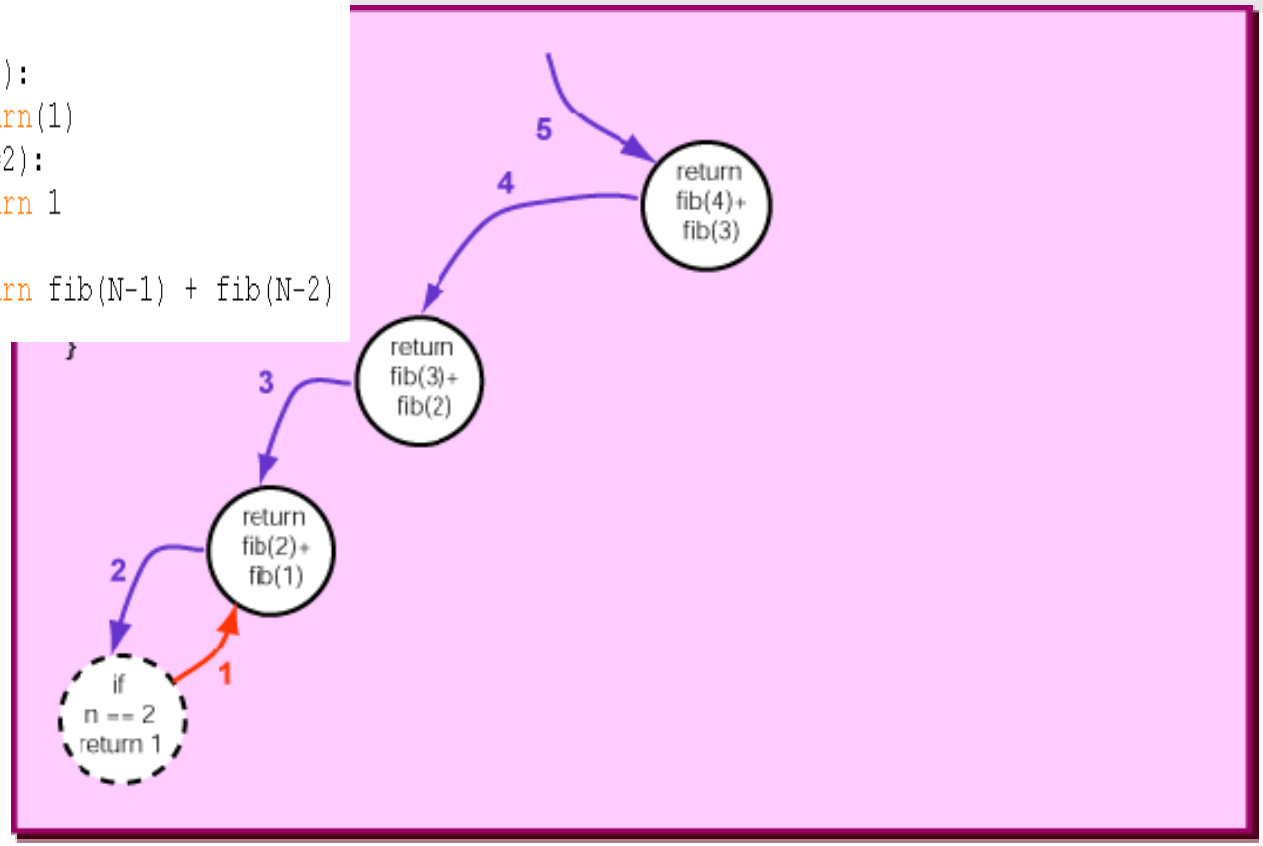
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



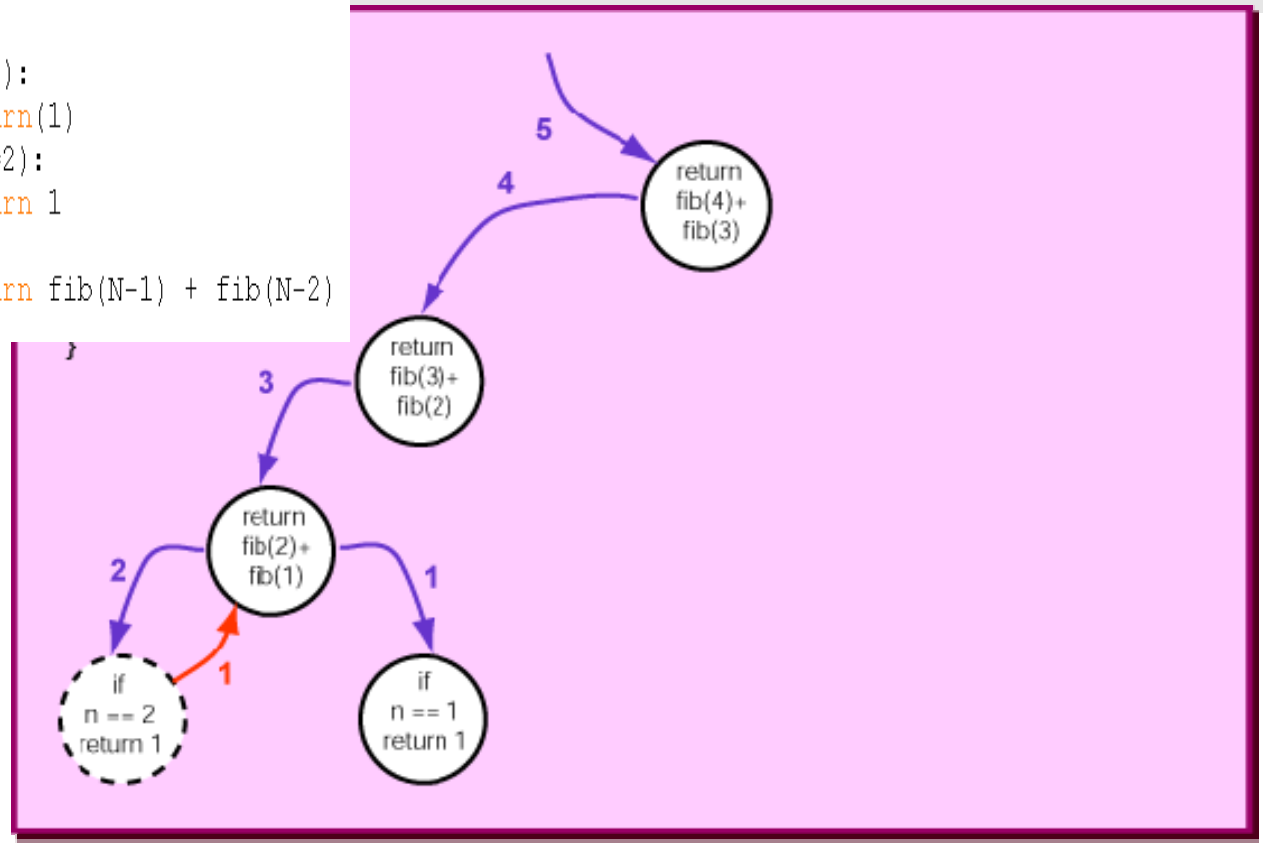
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



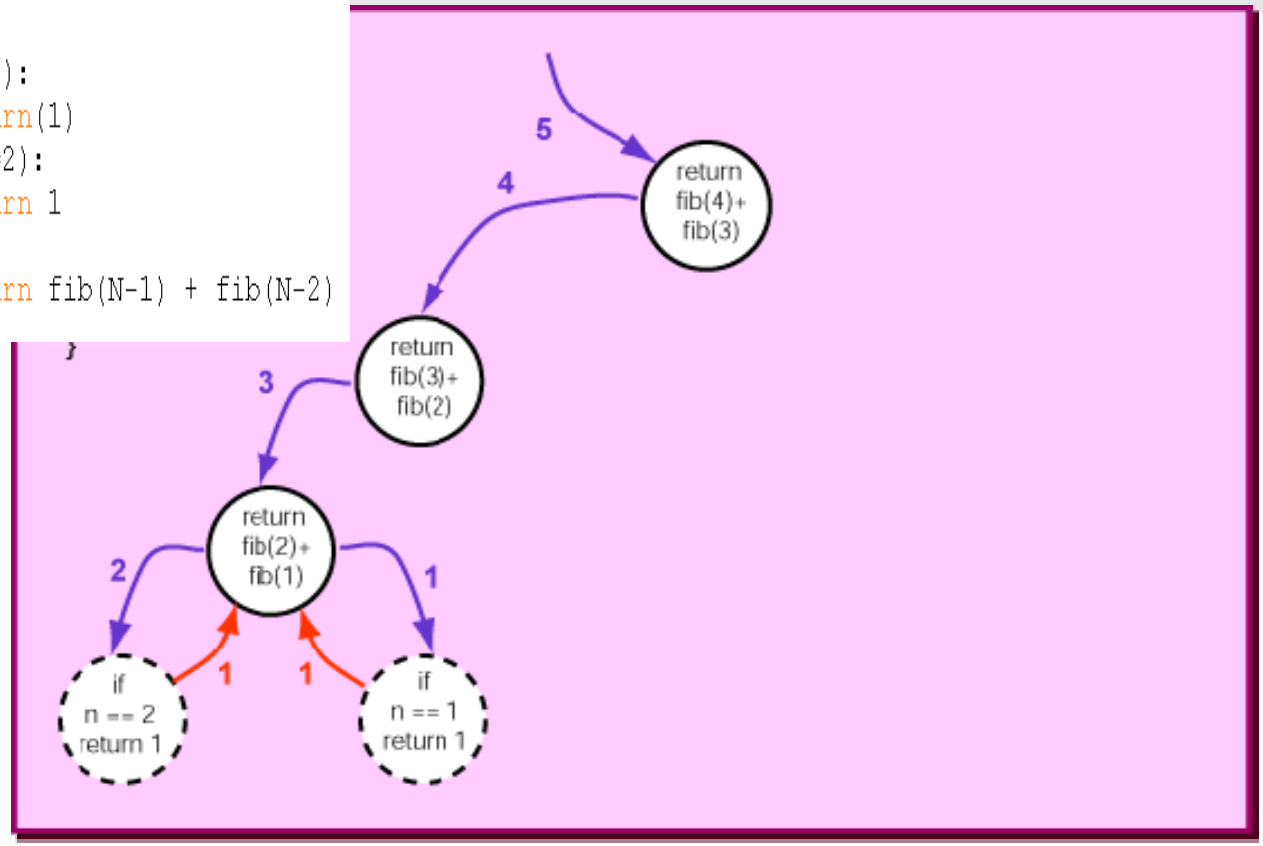
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



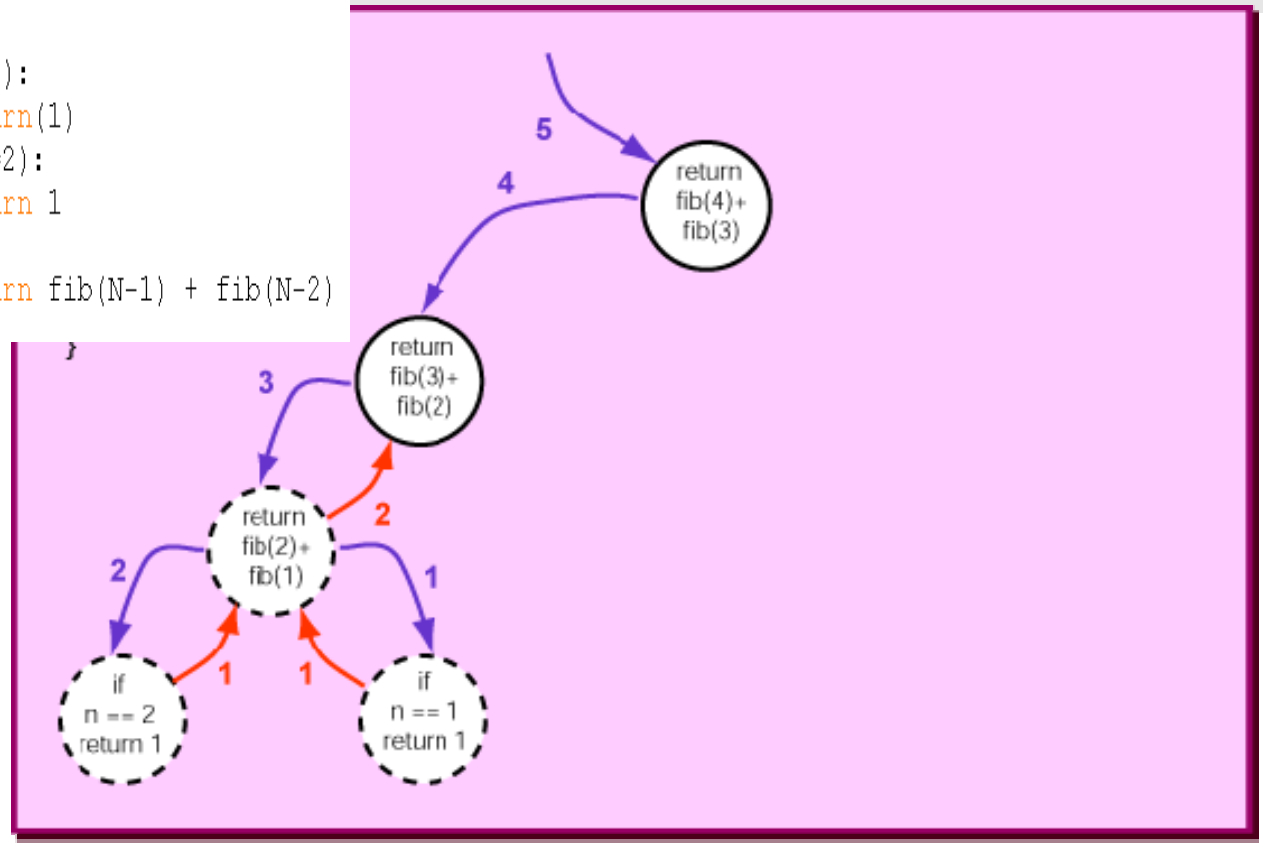
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



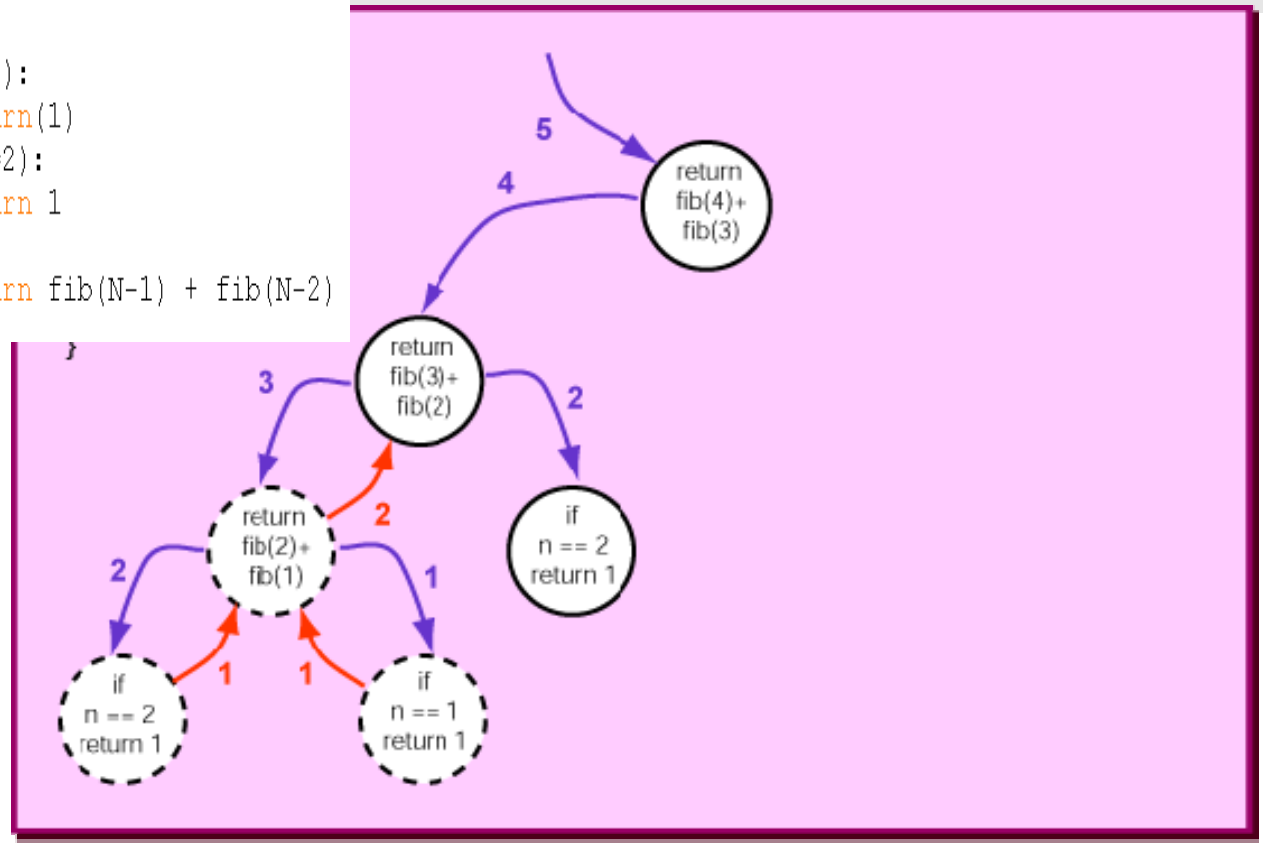
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



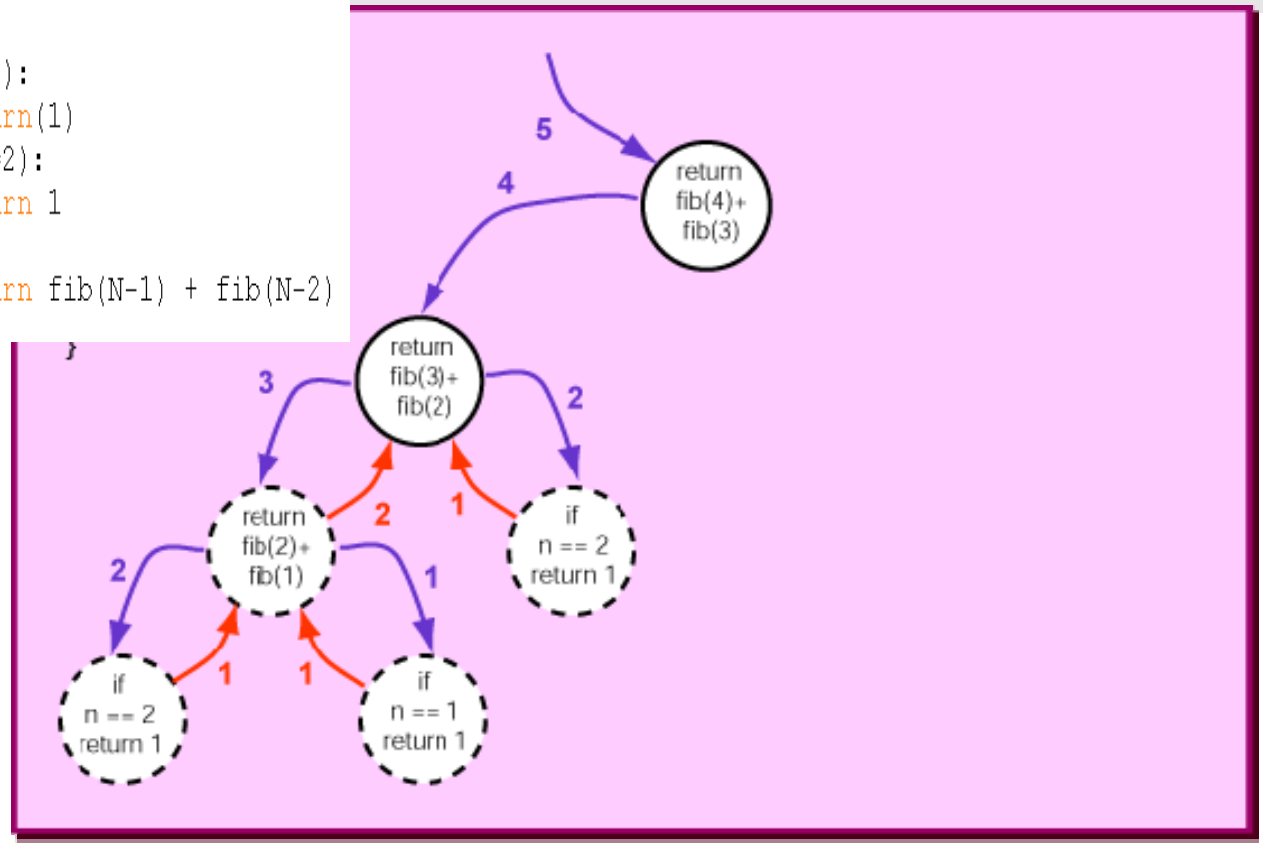
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



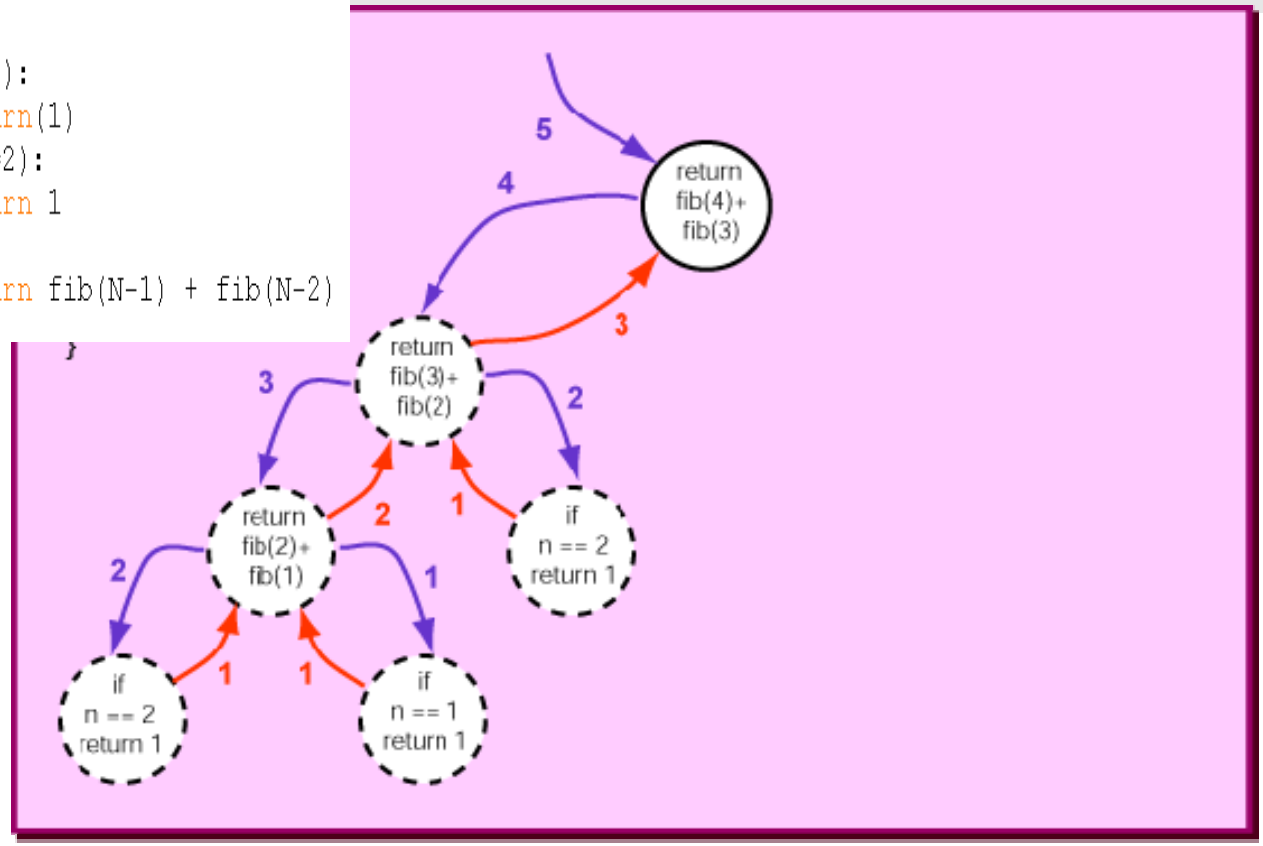
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



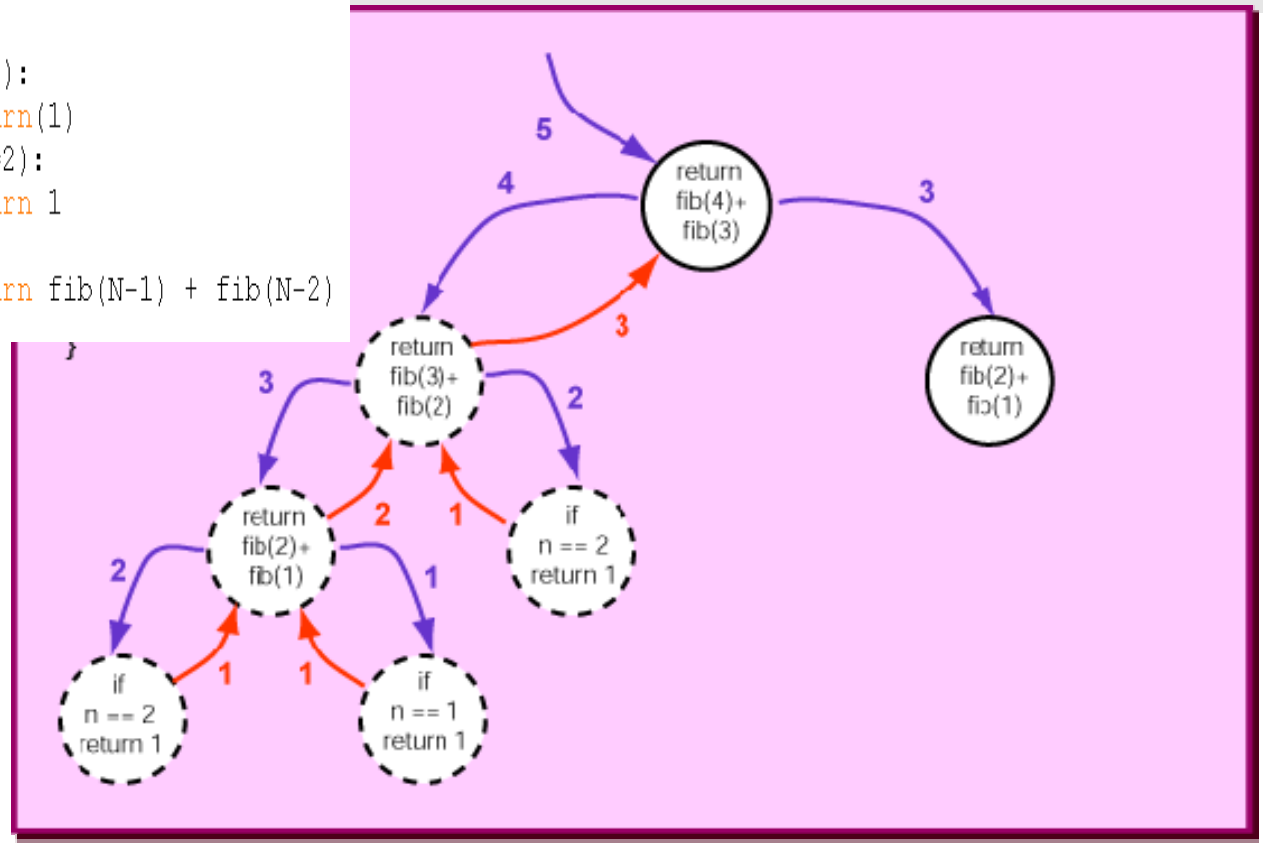
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



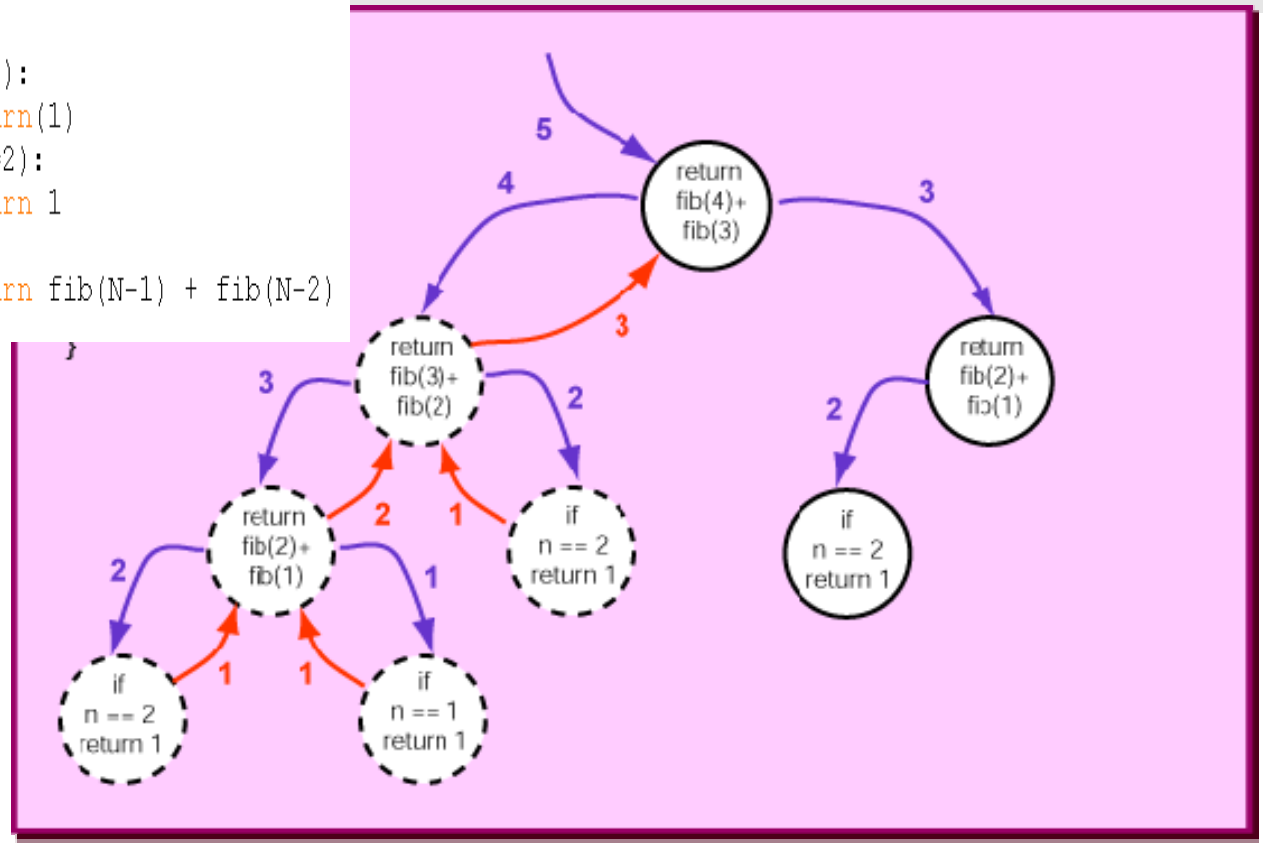
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



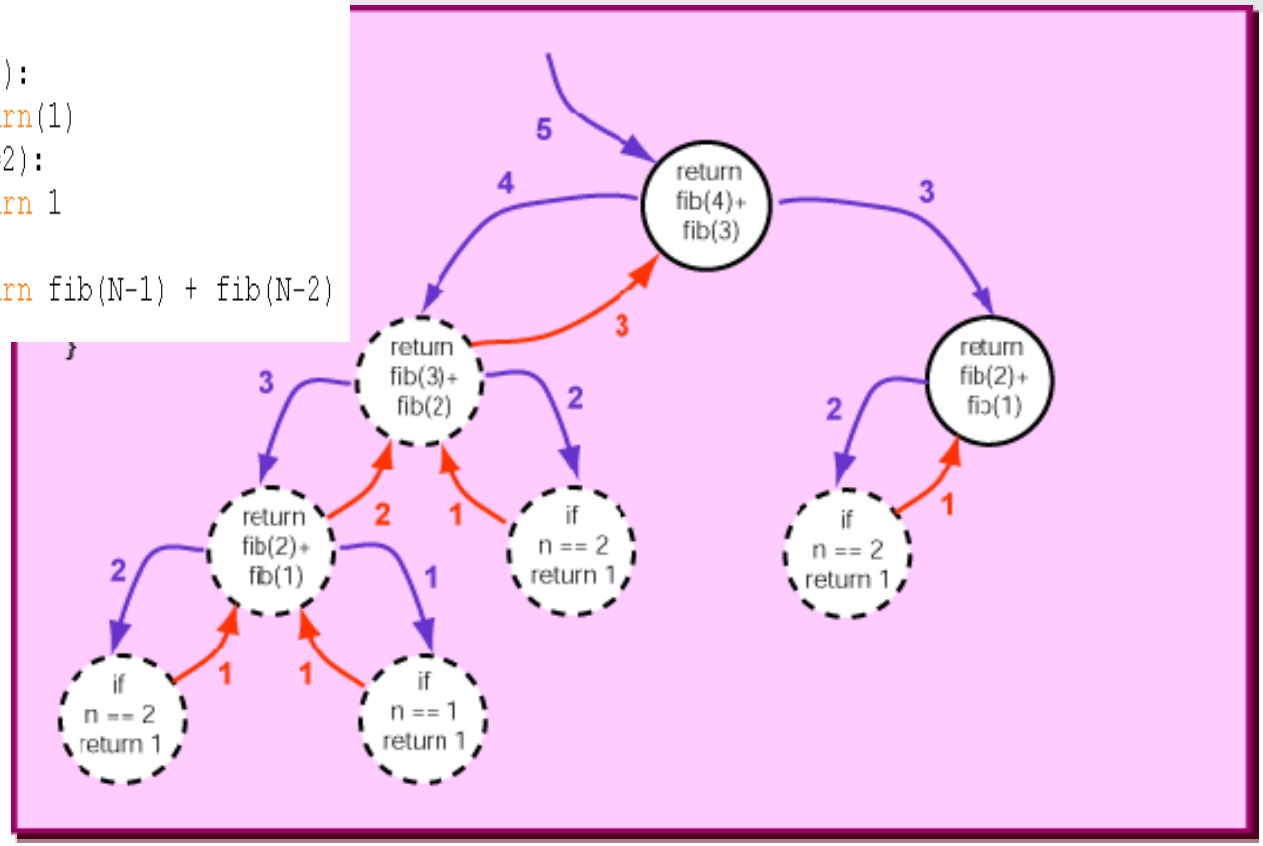
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



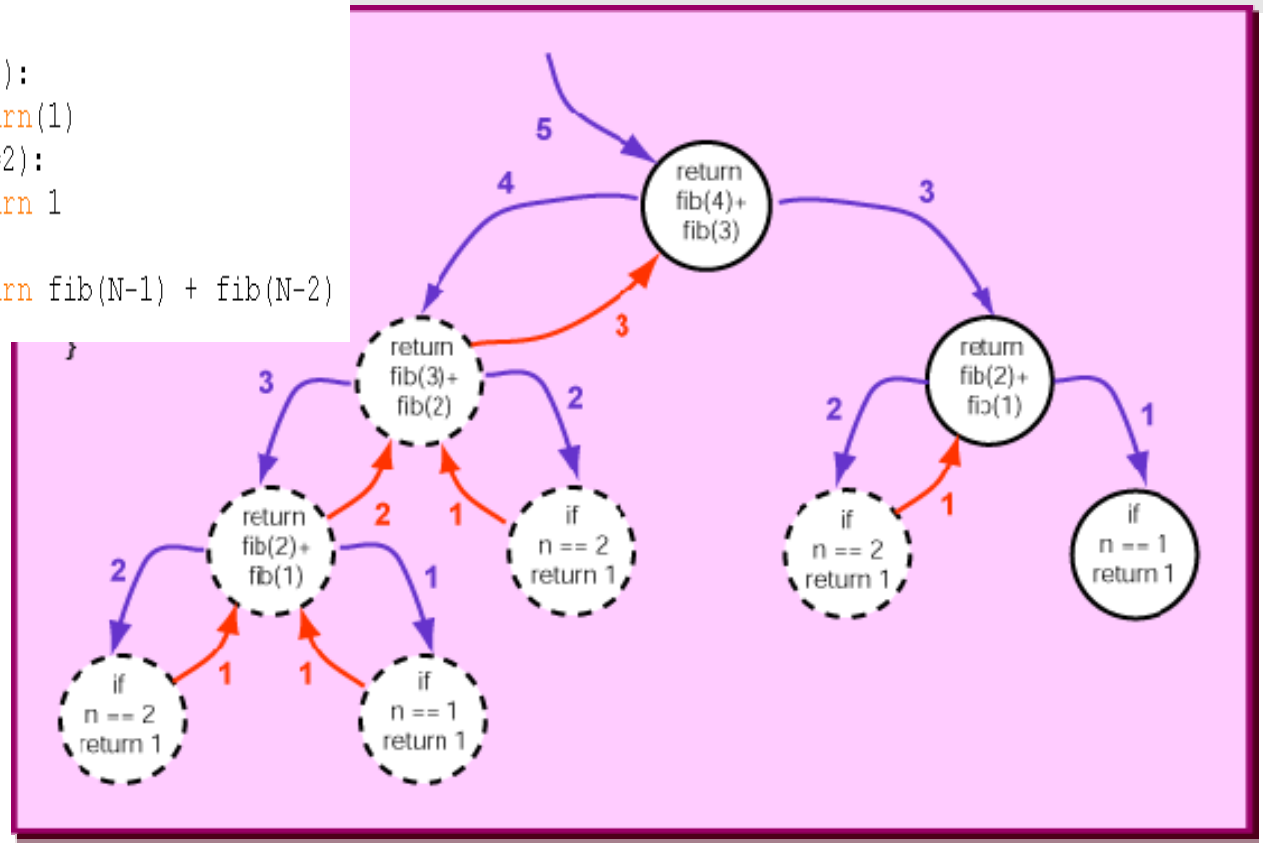
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



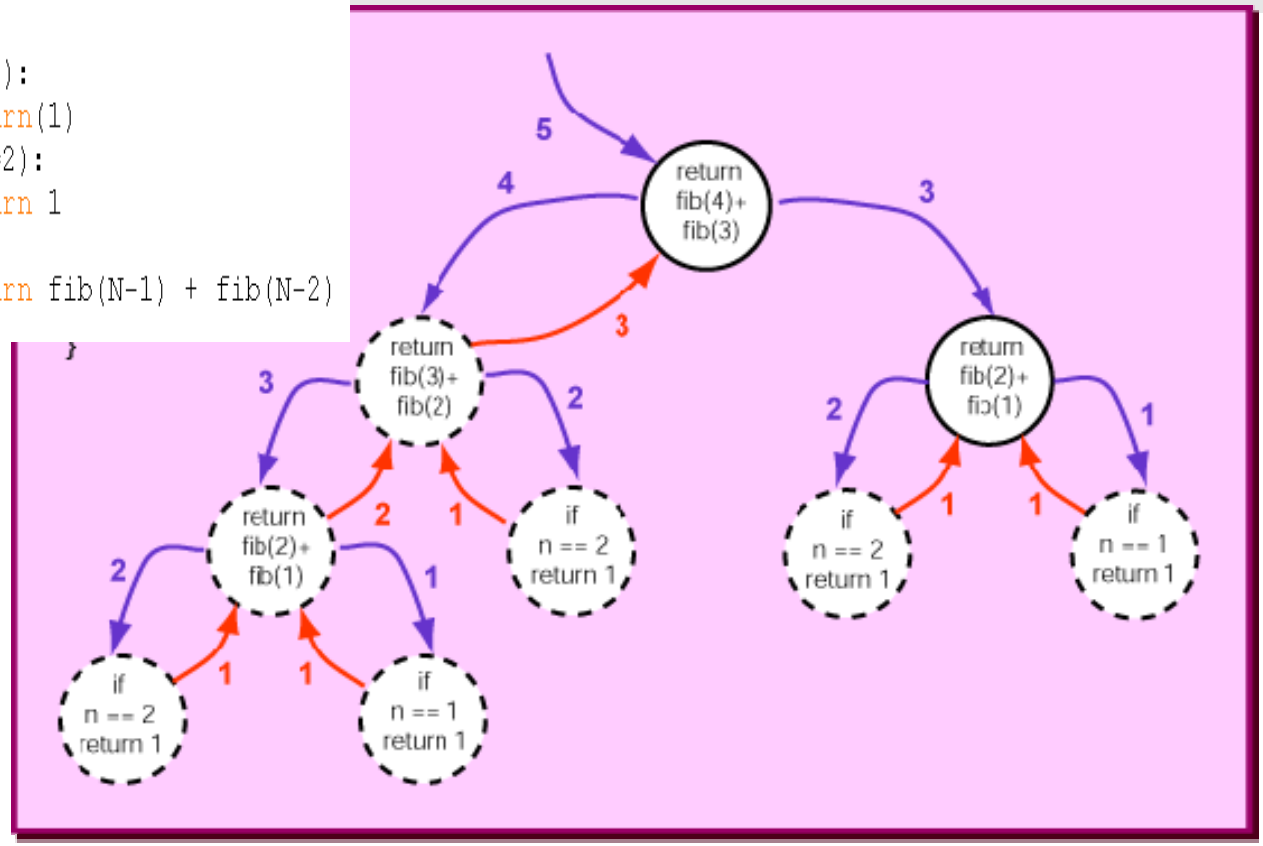
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



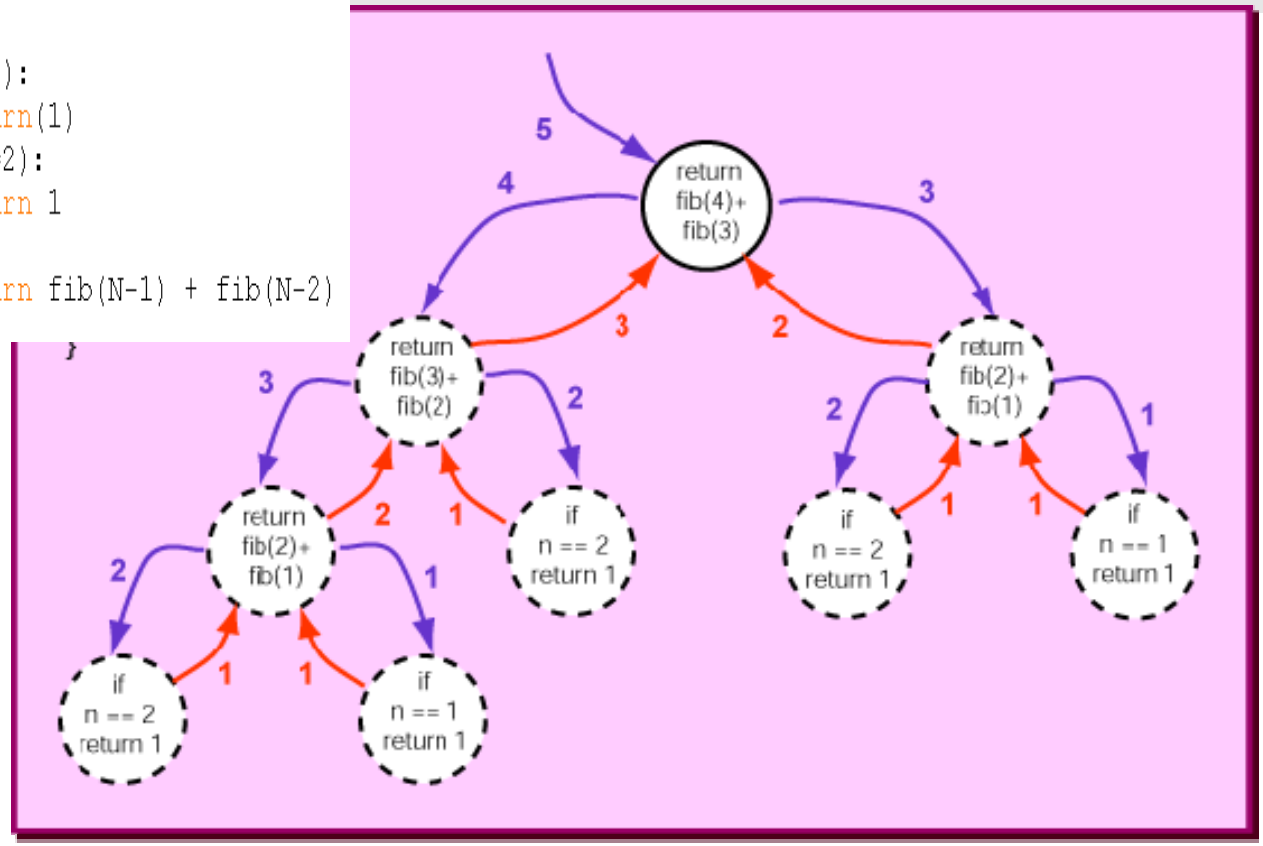
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



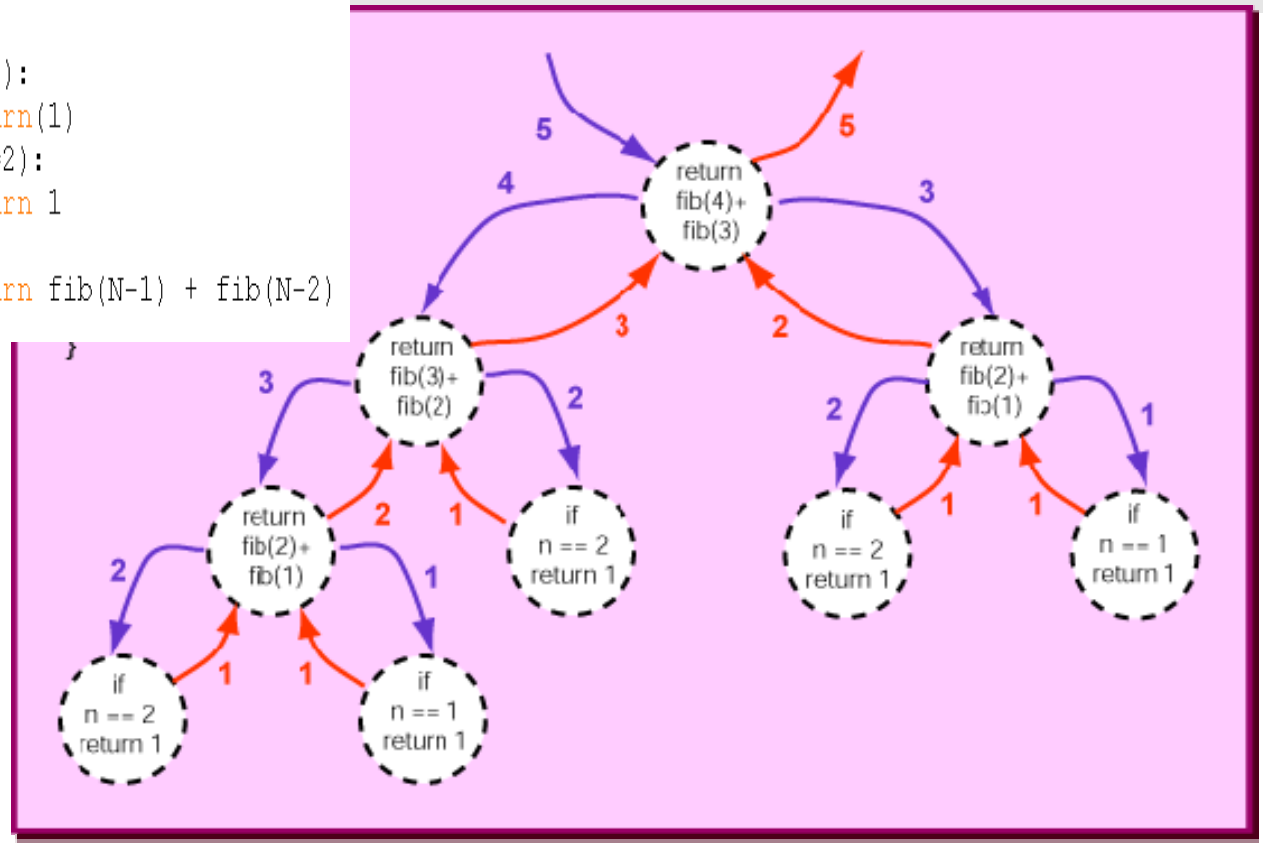
```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```

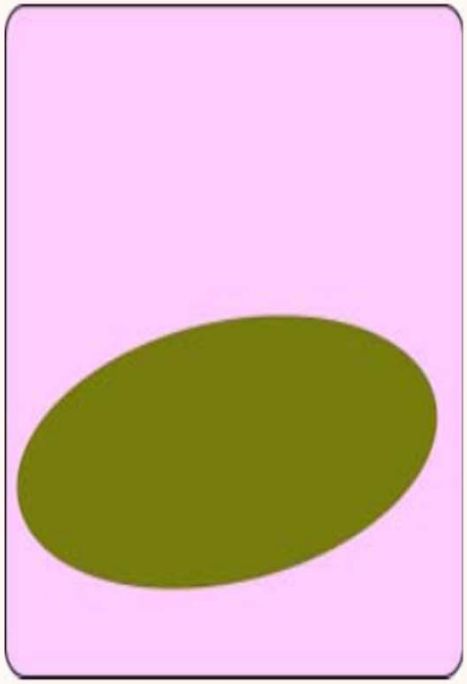


```
def fib(N):
    if (N==1):
        return(1)
    elif(N==2):
        return 1
    else:
        return fib(N-1) + fib(N-2)
```



Pyramidal Numbers

Let us use three-sided pyramids (sometimes called tetrahedrons). To make the pyramid, start with a triangular arrangement of tennis balls for the bottom layer. Next stack up tennis balls layer at a time, as at right.

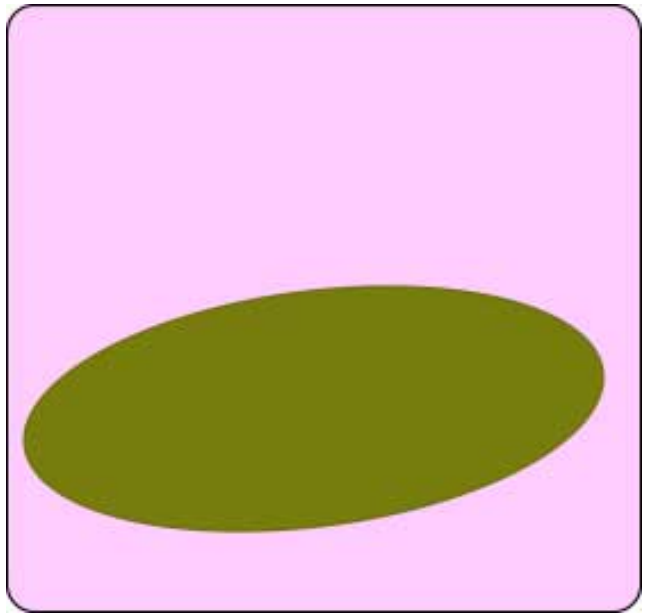


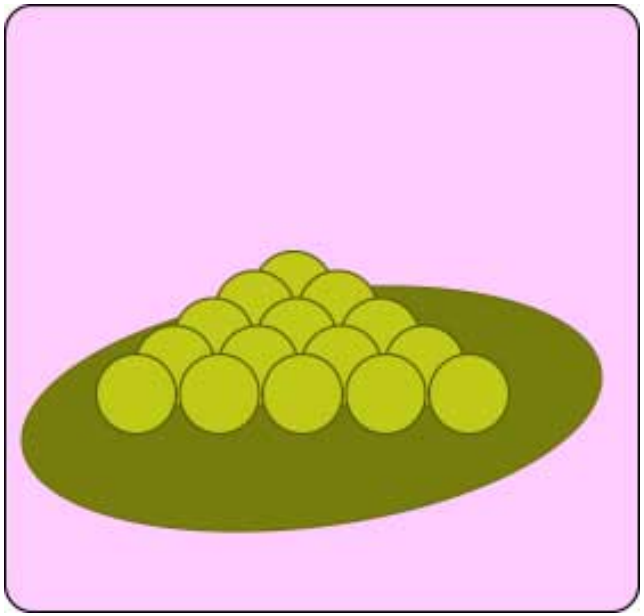
If the base of the pyramid has 5 balls along a side, how many balls does the entire pyramid have in it? Let us call this a **pyramidal number**.

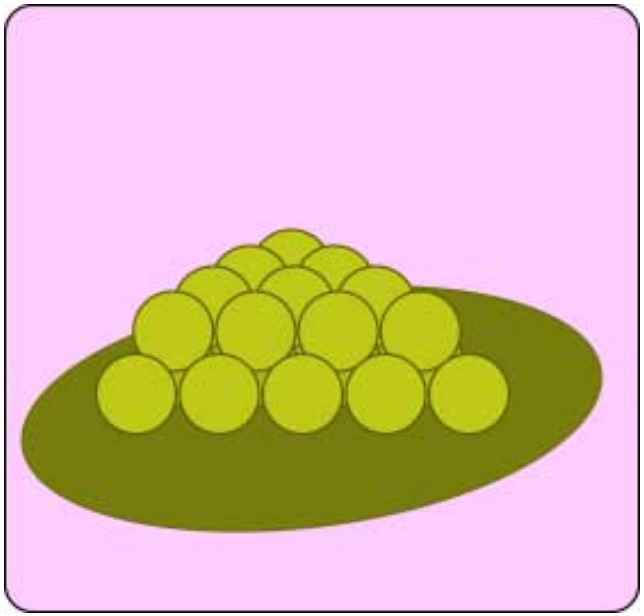
Here is a chart that shows the number of balls in the side of a base, **N**, and the corresponding pyramidal numbers.

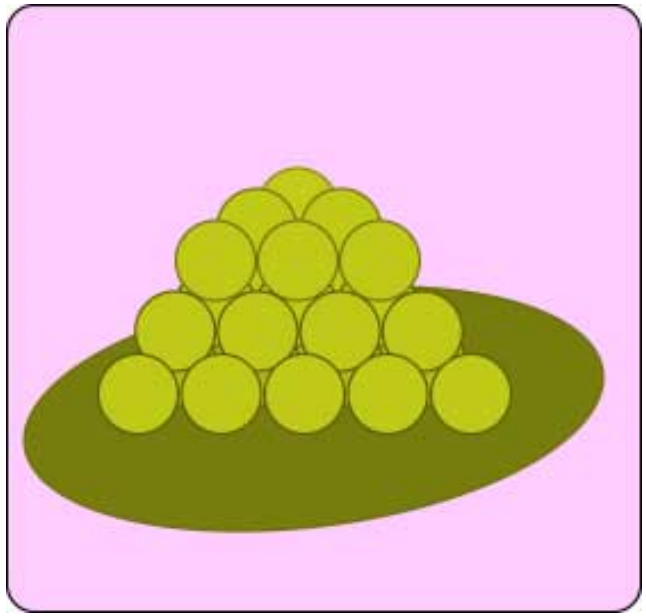
N	1	2	3	4	5	6	7
Pyramid(N)	1						

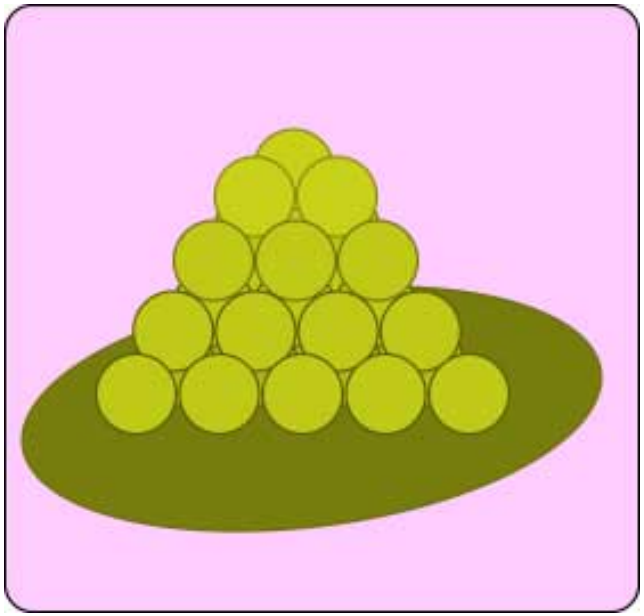
The picture starts with the base of the pyramid for **Pyramid(5)**. Click a few times to see the complete pyramid. Smaller pyramids can be found within this one. For small pyramids, count the balls within them. For larger pyramids, try to clever way to calculate the number of balls.

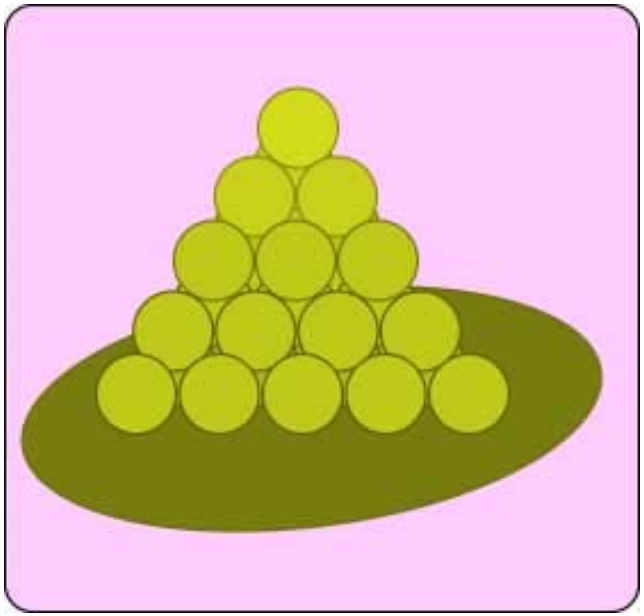






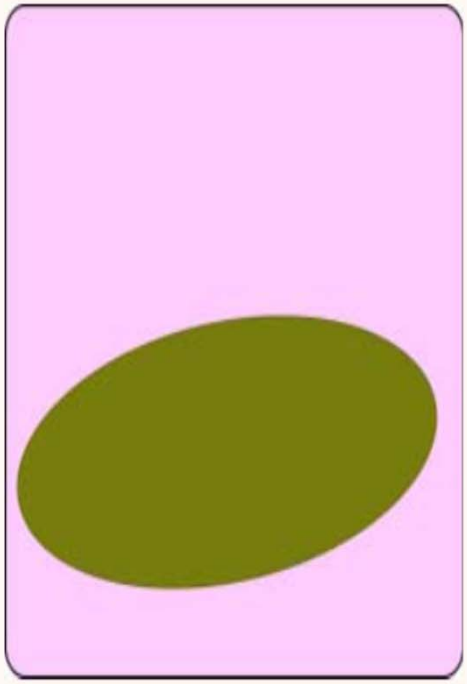






Pyramidal Numbers

Let us use three-sided pyramids (sometimes called tetrahedrons). To make the pyramid, start with a triangular arrangement of tennis balls for the bottom layer. Next stack up tennis balls layer at a time, as at right.



If the base of the pyramid has 5 balls along a side, how many balls does the entire pyramid have in it? Let us call this a **pyramidal number**.

Here is a chart that shows the number of balls in the side of a base, **N**, and the corresponding pyramidal numbers.

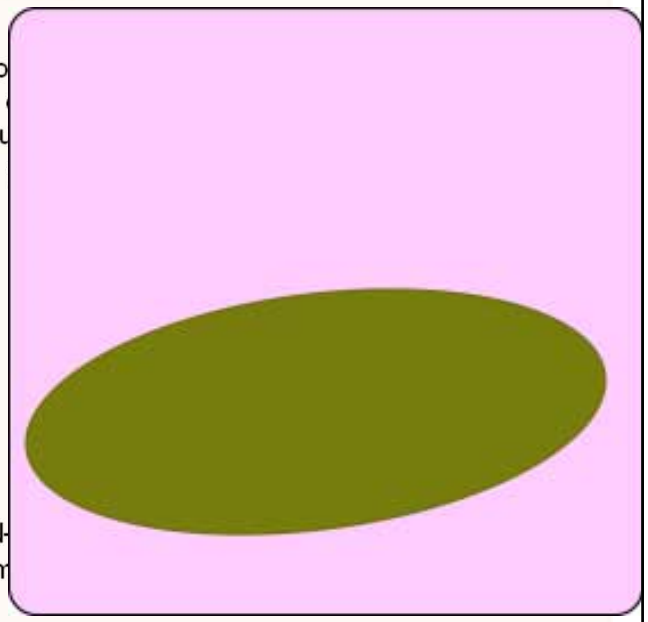
N	1	2	3	4	5	6	7
Pyramid(N)	1						

The picture starts with the base of the pyramid for **Pyramid(5)**. Click a few times to see the complete pyramid. Smaller pyramids can be found within this one. For small pyramids, count the balls within them. For larger pyramids, try to clever way to calculate the number of balls.

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to kno
how many balls are in each layer of the pyramid. Of course, the number
the balls in the pyramid is the sum of the number of balls in each layer. Bu
the number of balls in layer N is `Triangle(N)`, the number of balls
a triangle with a side of N balls.

N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84



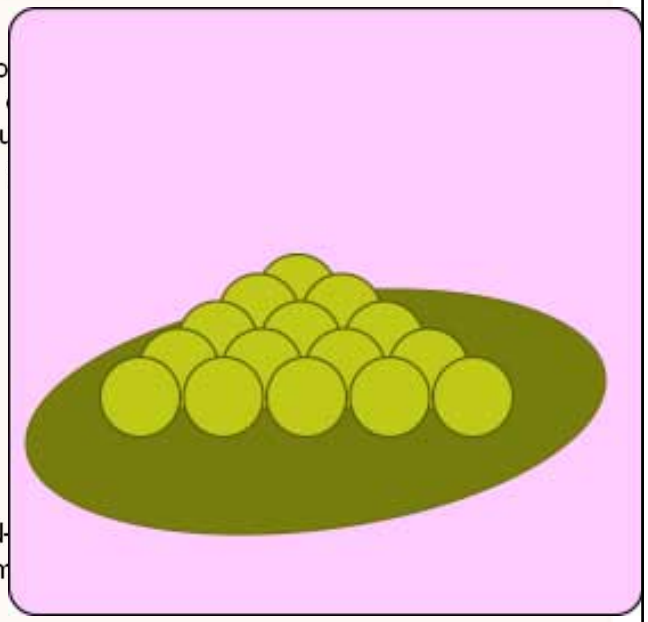
Say that you know that there are `Pyramid(N-1)` balls in the first N-
layers (counting from the top of the pyramid). Then the following schem
looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to kno
how many balls are in each layer of the pyramid. Of course, the number
the balls in the pyramid is the sum of the number of balls in each layer. Bu
the number of balls in layer N is `Triangle(N)`, the number of balls
a triangle with a side of N balls.

N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84



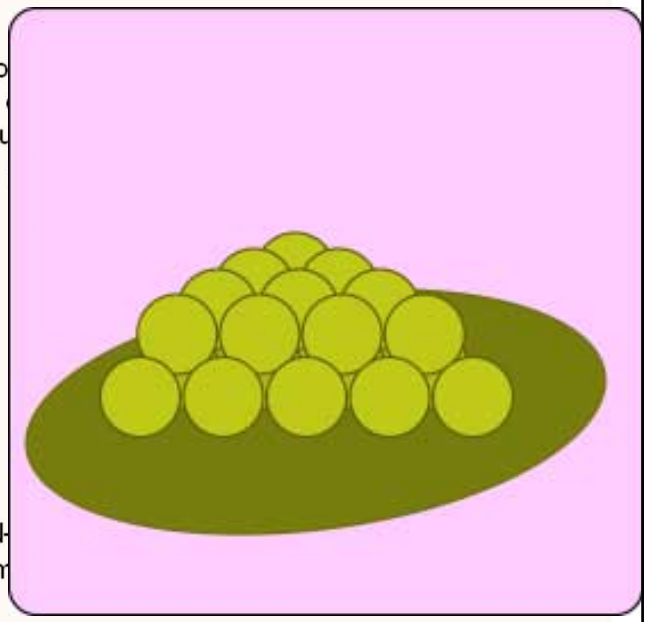
Say that you know that there are `Pyramid(N-1)` balls in the first N-
layers (counting from the top of the pyramid). Then the following schem
looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to kno
how many balls are in each layer of the pyramid. Of course, the number
the balls in the pyramid is the sum of the number of balls in each layer. Bu
the number of balls in layer N is `Triangle(N)`, the number of balls
a triangle with a side of N balls.

N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84



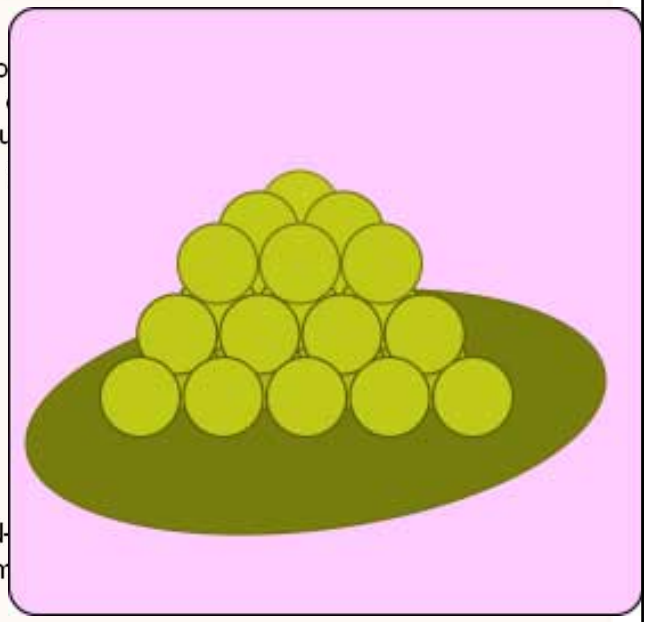
Say that you know that there are `Pyramid(N-1)` balls in the first N-
layers (counting from the top of the pyramid). Then the following schem
looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to kno
how many balls are in each layer of the pyramid. Of course, the number
the balls in the pyramid is the sum of the number of balls in each layer. Bu
the number of balls in layer N is `Triangle(N)`, the number of balls
a triangle with a side of N balls.

N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84

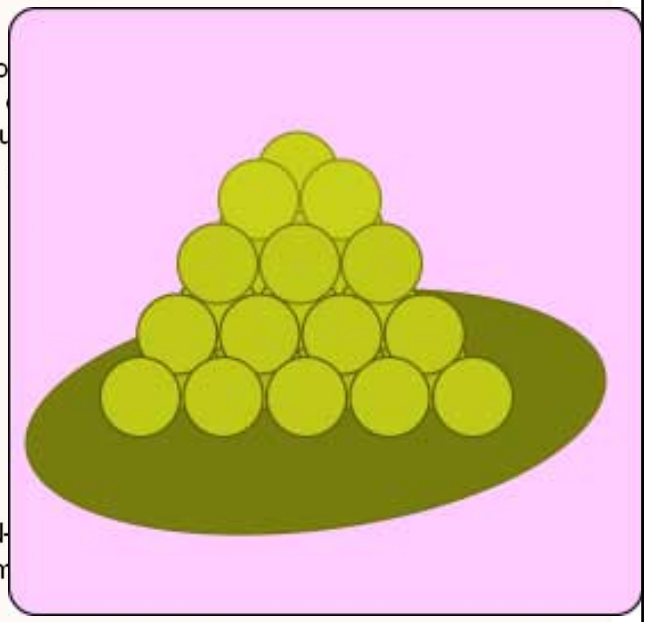


Say that you know that there are `Pyramid(N-1)` balls in the first N-
layers (counting from the top of the pyramid). Then the following schem
looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to kno
how many balls are in each layer of the pyramid. Of course, the number
the balls in the pyramid is the sum of the number of balls in each layer. Bu
the number of balls in layer N is `Triangle(N)`, the number of balls
a triangle with a side of N balls.



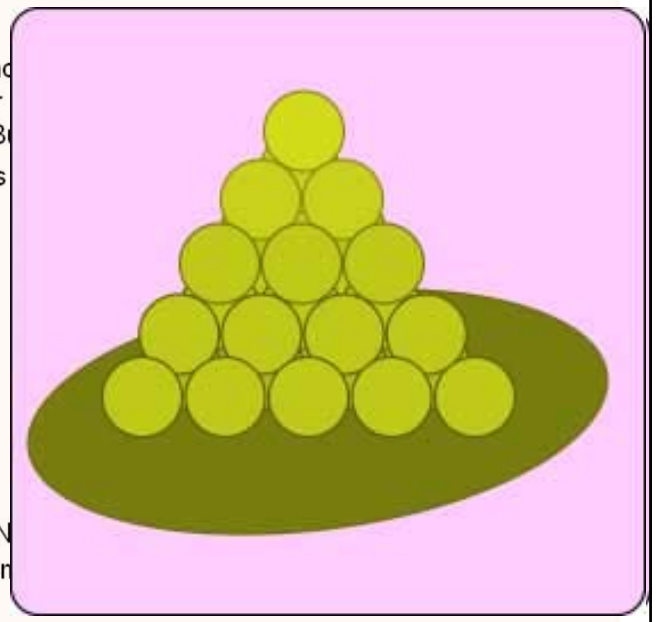
N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84

Say that you know that there are `Pyramid(N-1)` balls in the first N-
layers (counting from the top of the pyramid). Then the following schem
looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to kn
how many balls are in each layer of the pyramid. Of course, the number
the balls in the pyramid is the sum of the number of balls in each layer. B
the number of balls in layer N is `Triangle(N)`, the number of balls
a triangle with a side of N balls.



N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84

Say that you know that there are `Pyramid(N-1)` balls in the first N
layers (counting from the top of the pyramid). Then the following schen
looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Base Case of the Pyramid

A recursive definition (or a recursive algorithm) needs two parts:

1. If the problem is easy, solve it immediately.
2. If the problem can't be solved immediately, divide it into smaller problems, then:
    - Solve the smaller problems by applying this procedure to each of them.

In terms of pyramidal numbers, this is:

```
Pyramid(1) = 1
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Oddly, the base case for pyramidal numbers is the peak of the pyramid.

Definition that uses another Function

As long as everything in the definition for `Pyramid()` is defined someplace, everything is fine

Here (for reference) is `Pyramid()`:

```
Pyramid(1) = 1
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

And here (for review) is `Triangle()`:

```
Triangle( 1 ) = 1
Triangle( N ) = N + Triangle( N-1 )
```

Given these two definitions (and, if you are picky, the definitions of addition and subtraction) `Pyramid()` is completely defined.

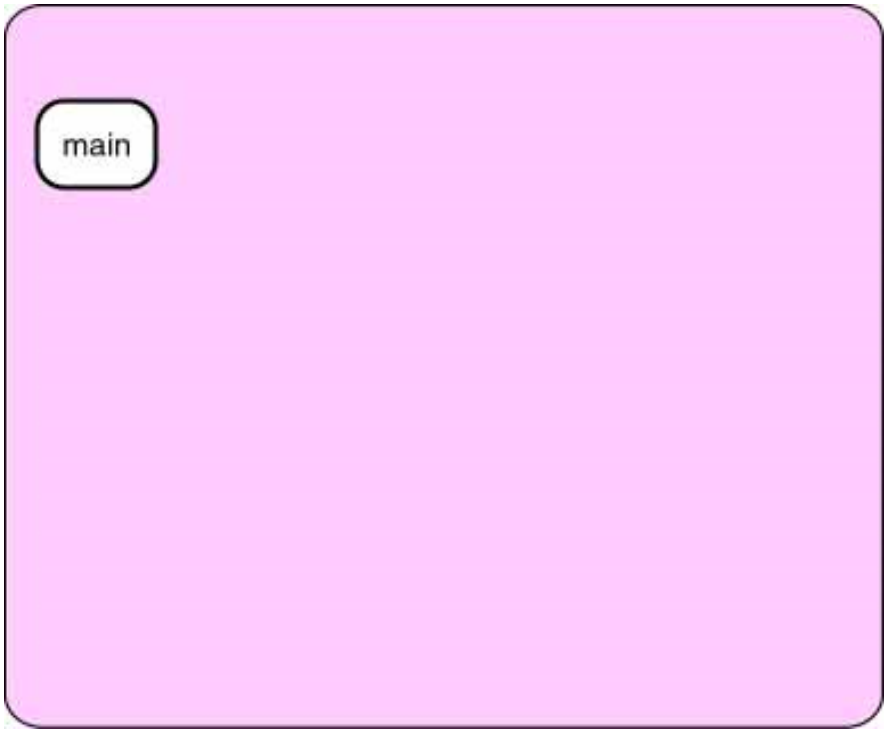
pyra.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\pyra.py (3.4.4)

File Edit Format Run Options Window Help

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)

def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)


n=int(input('Enter a number: '))
result=Pyramid(n)
print('Pyramid(', n, ') is: ', result)
```

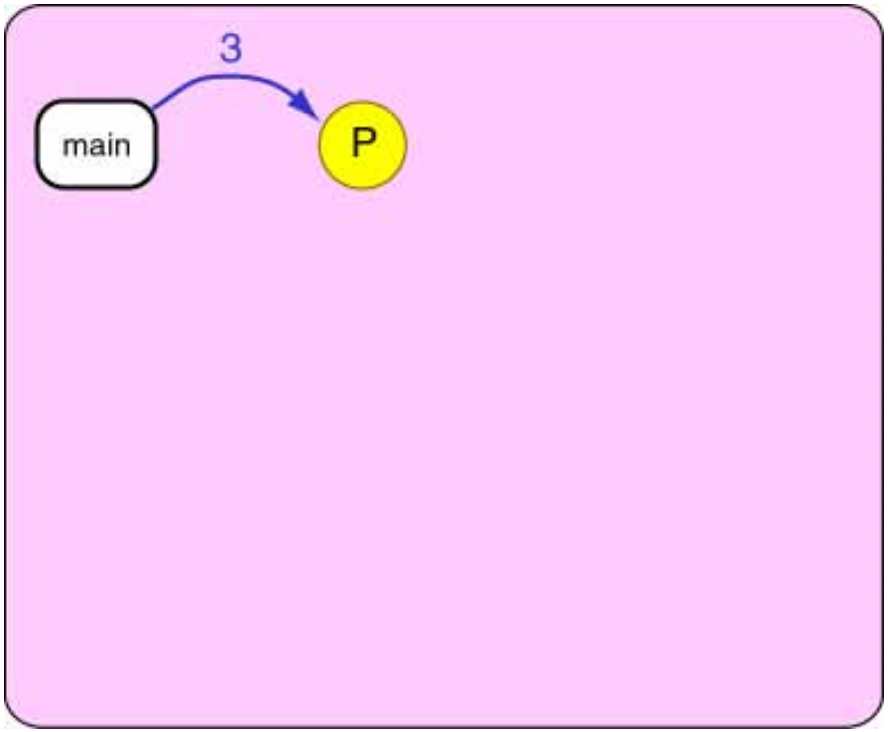
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



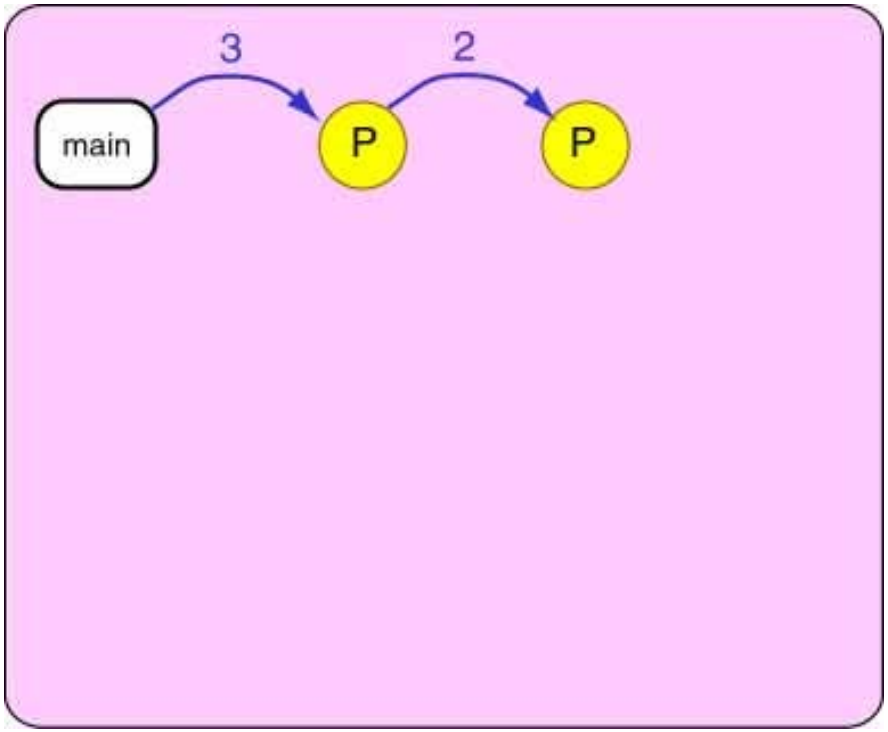
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



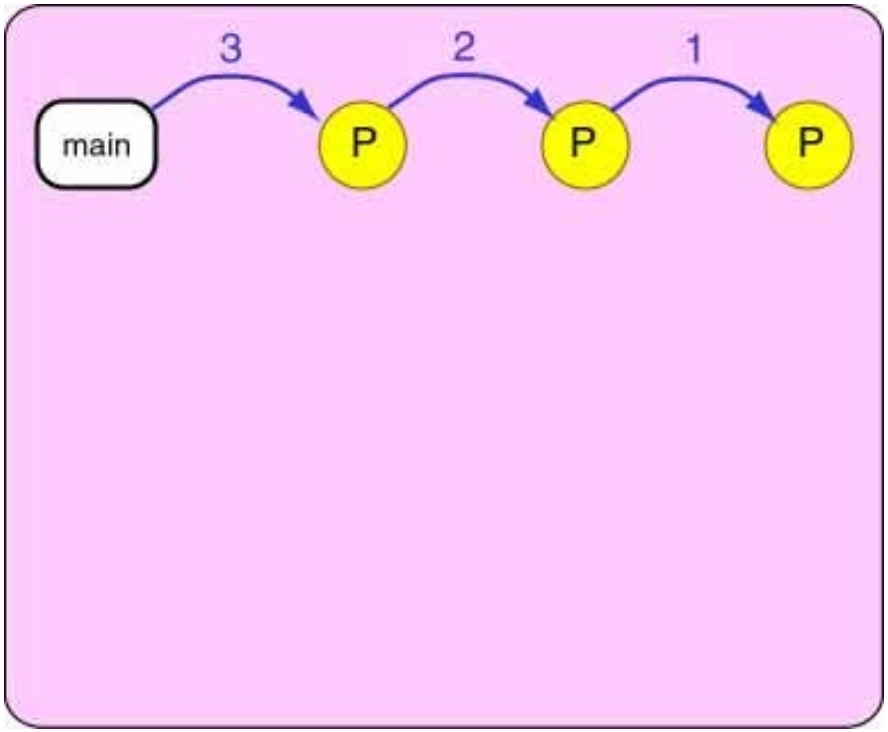
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



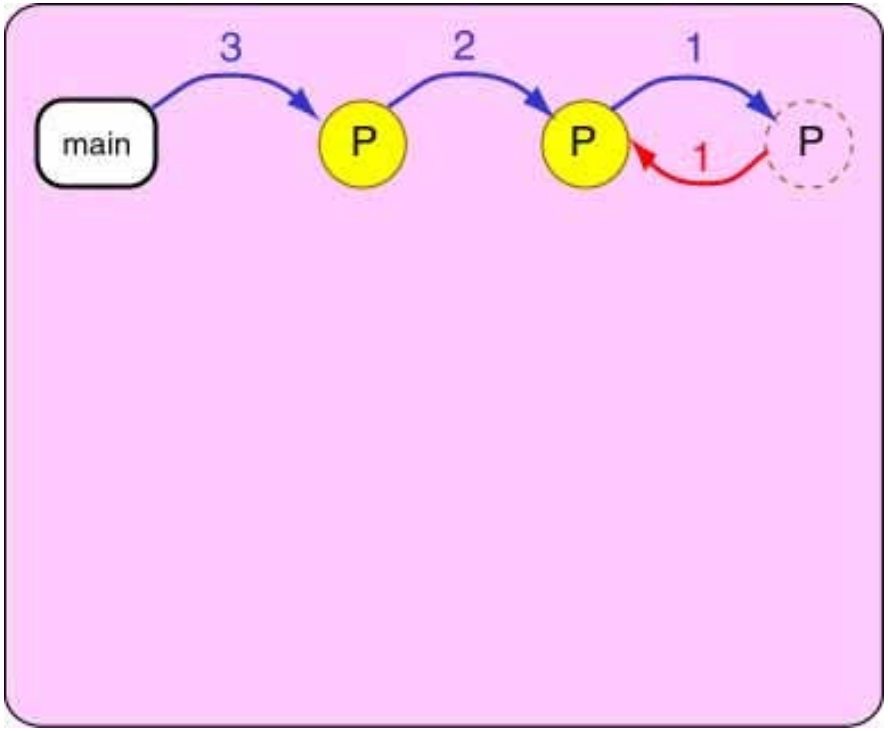
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



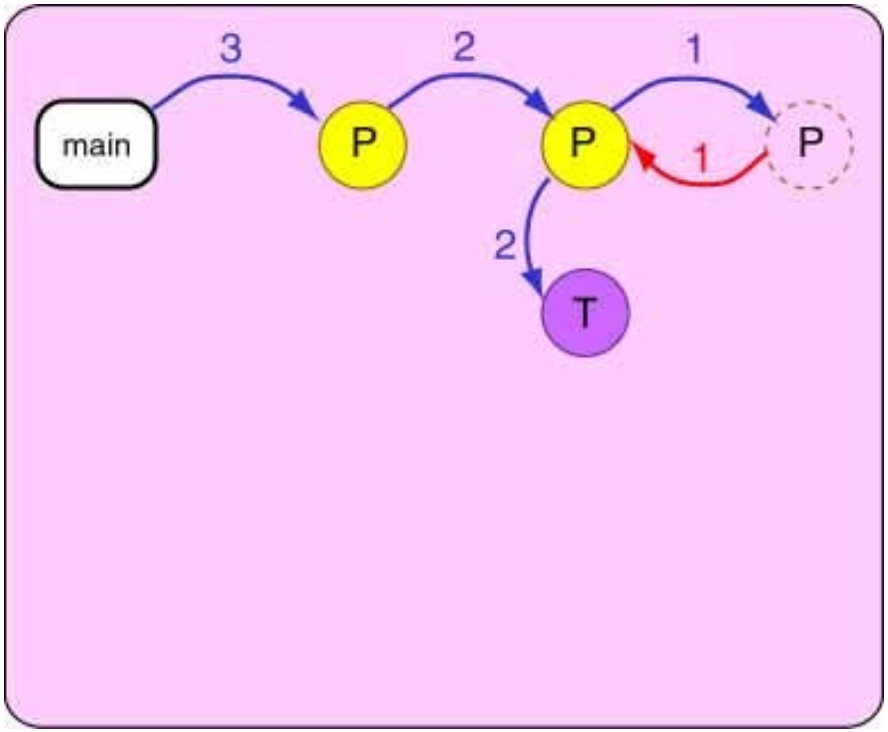
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



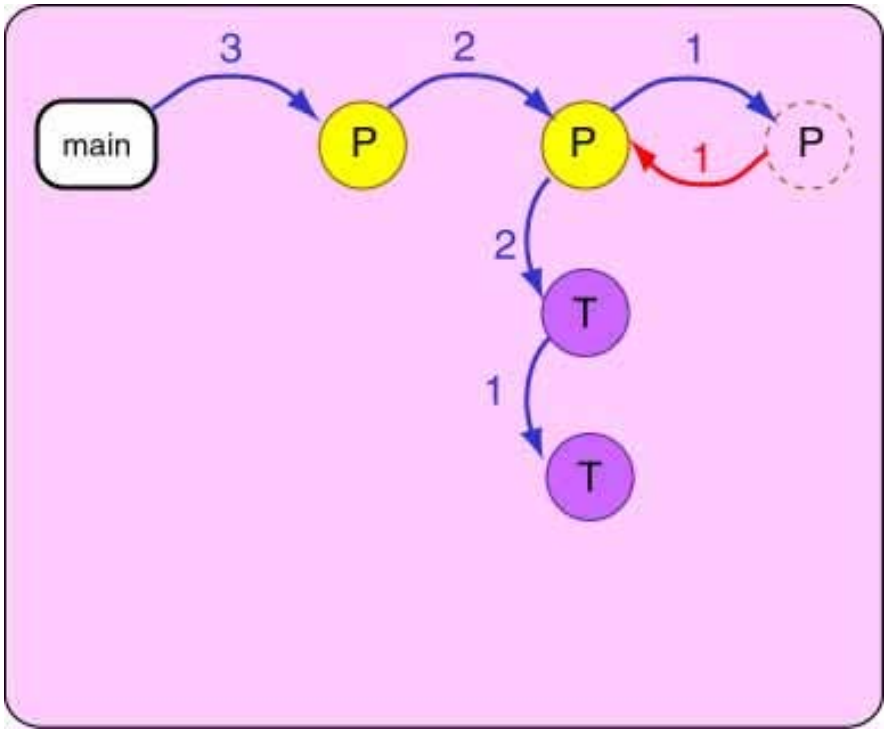
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



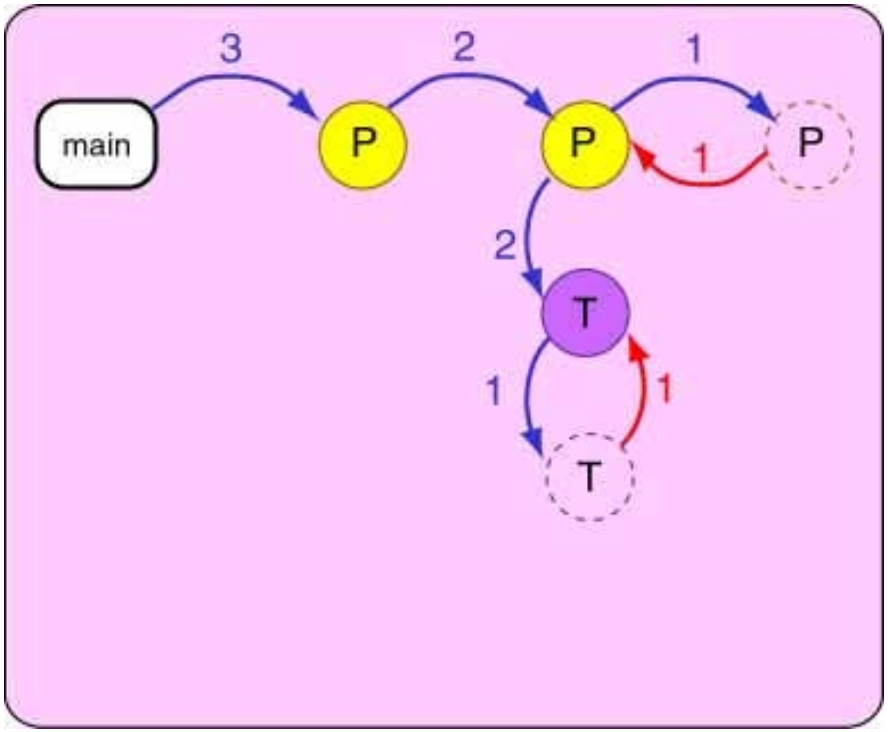
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



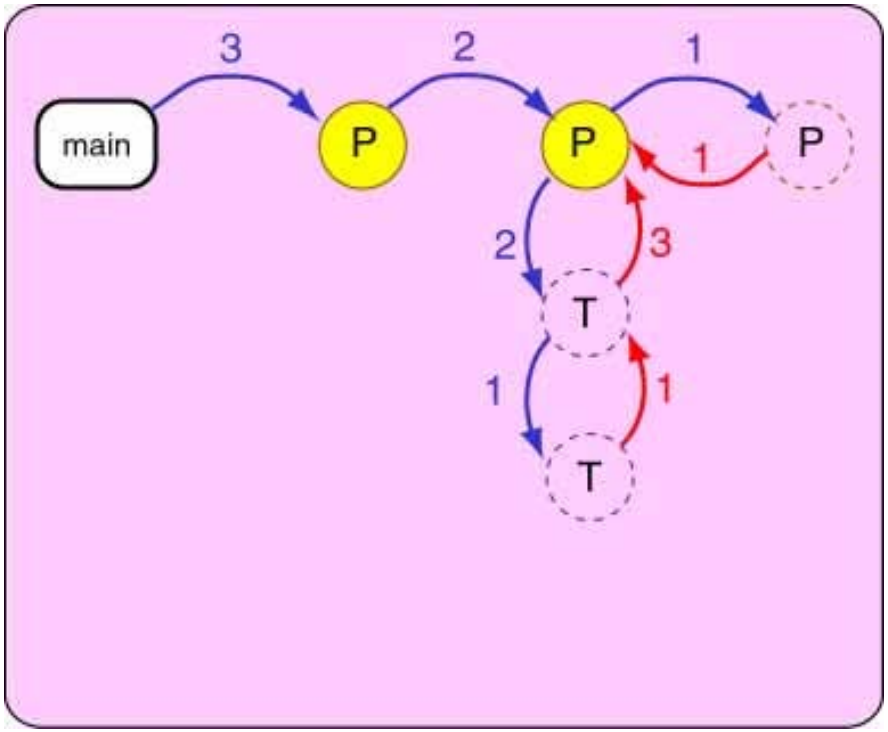
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



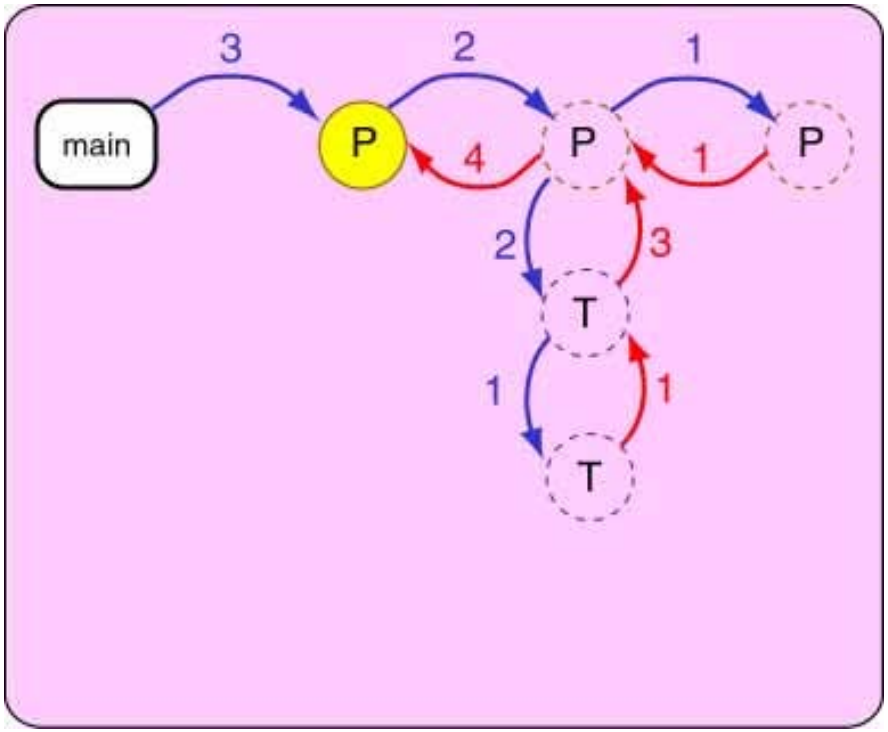
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



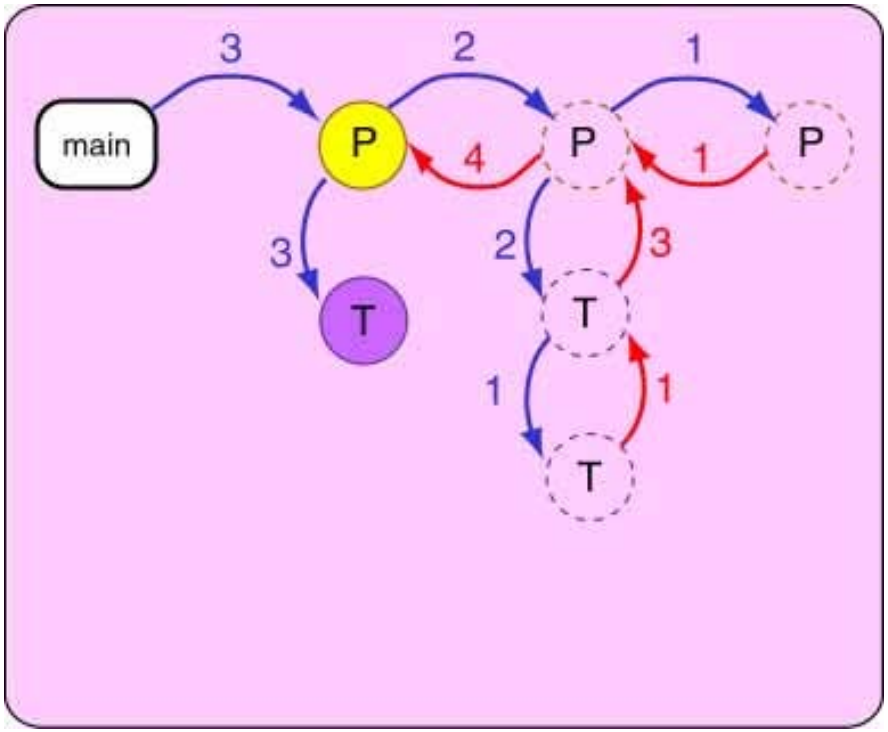
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



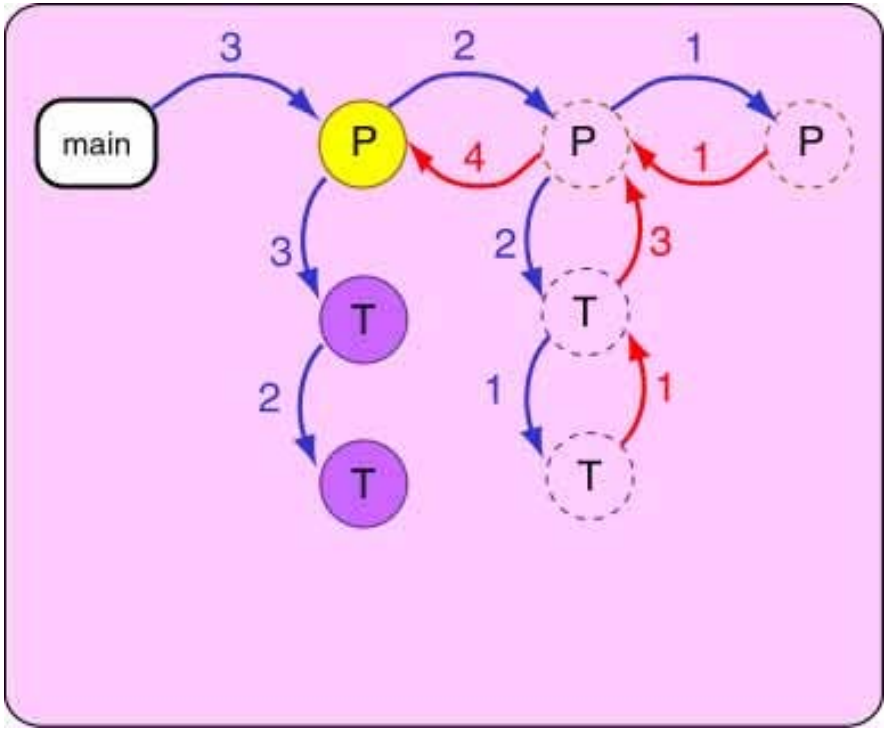
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



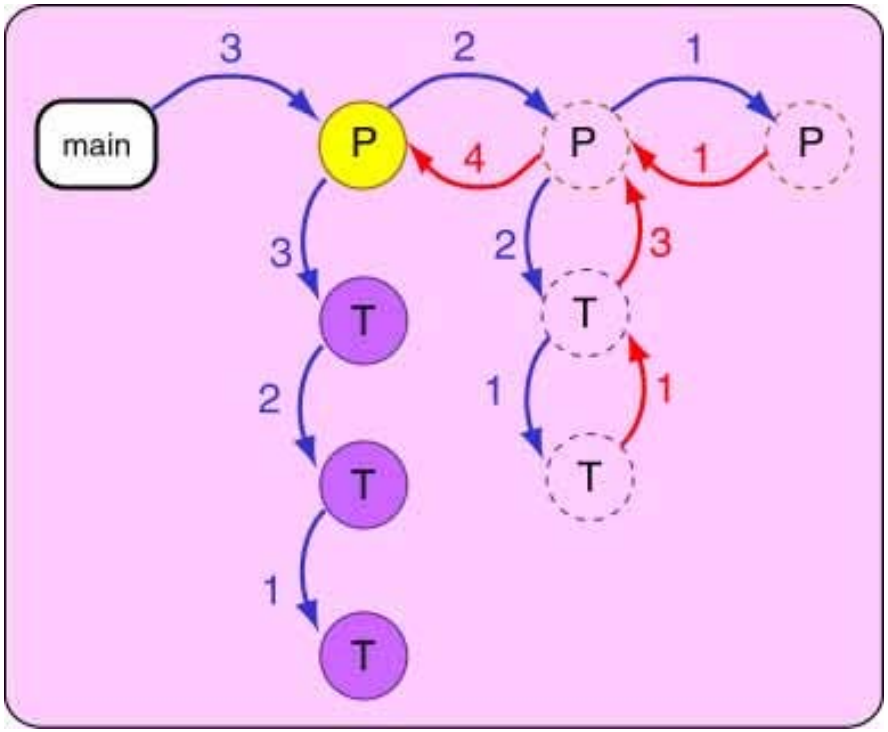
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



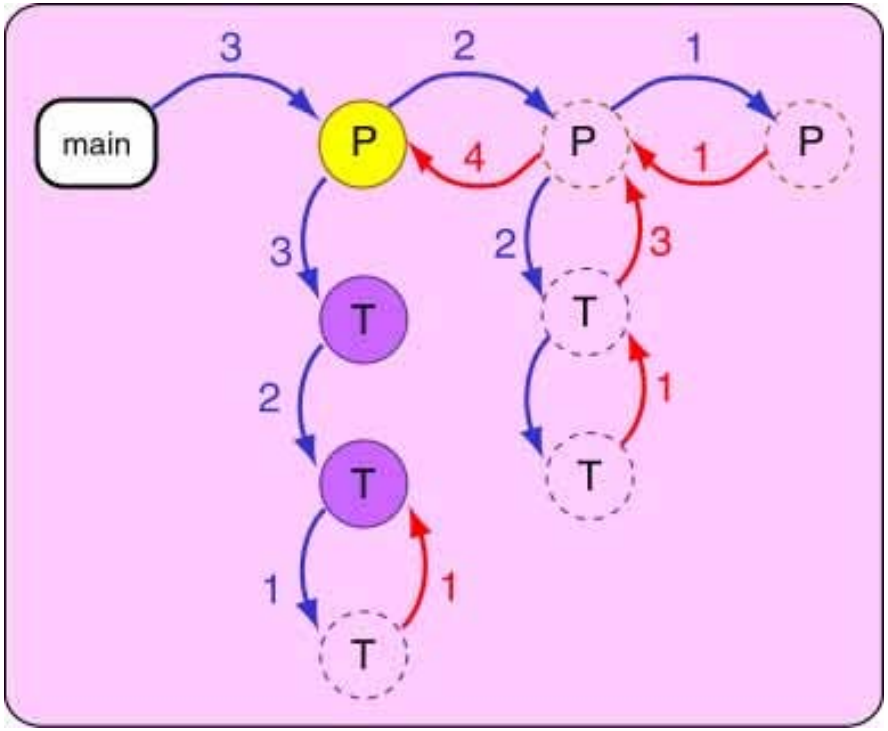
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



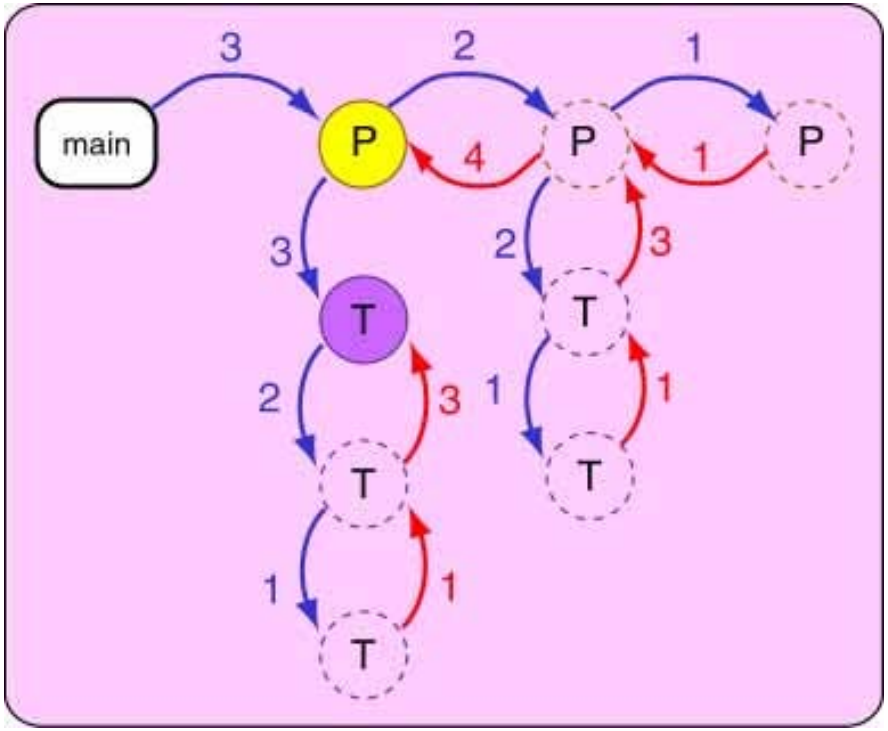
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



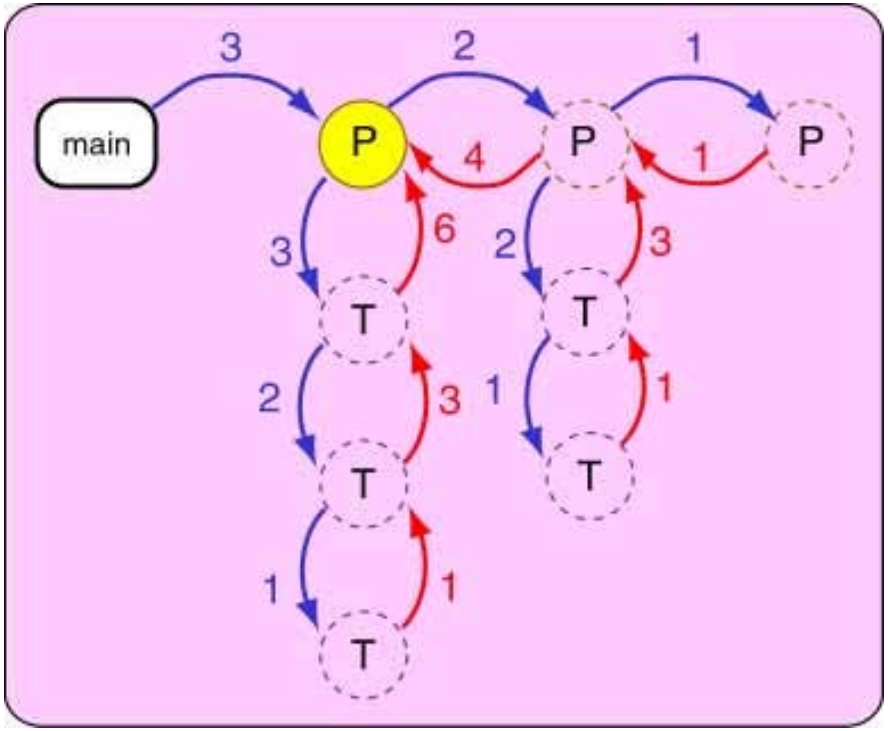
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



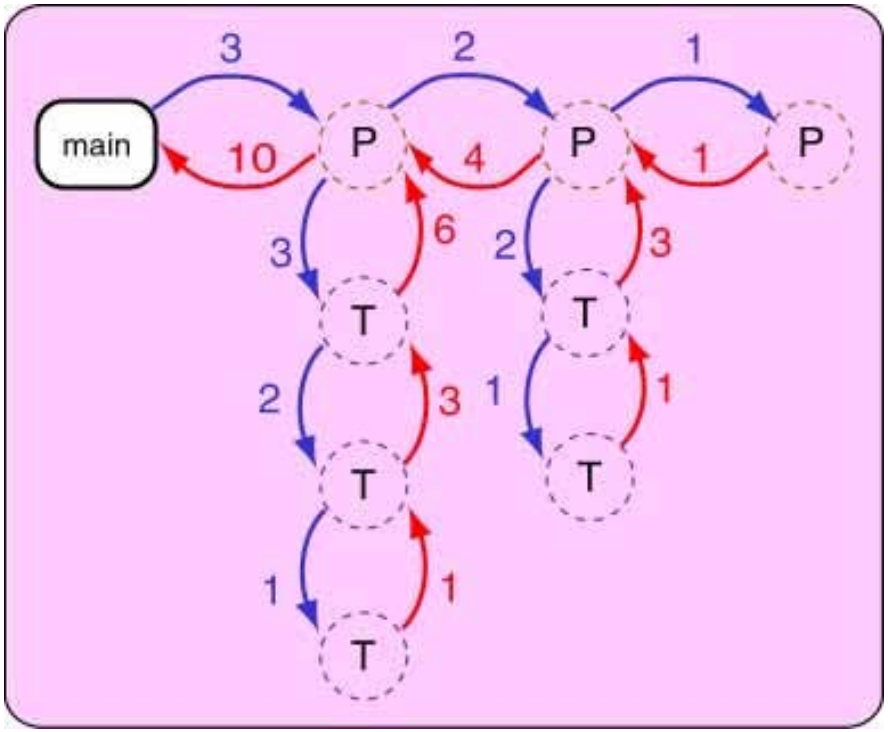
```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



```
def Pyramid(N):
    if (N==1):
        return 1
    else:
        return Pyramid(N-1) + Triangle(N)
```

```
def Triangle(N):
    if (N==1):
        return 1
    else:
        return N + Triangle(N-1)
```



End of Chapter