

Recursion

Idea: A function can call other functions, including itself!

Recursive functions are functions which call itself

Conceptually similar to a loop (because the same operation repeats each time the function calls itself)

Appropriate for “self-similar” problems

Analogy:

We all have parents, who themselves have parents, who themselves have parents, etc

Nested Russian dolls

Recursion example - building a tower with height 3

Iterative method

for each height from 1 to 3, place a block

Recursive method

to build a tower with height 3, first build a tower of height 2 and then add 1 block

How to design a recursive function

Define your **base case**

These are the cases where we won't recurse (e.g. the function does not call itself)

Prevents your recursion from running forever

Infinite recursion throws the error "maximum recursion depth reached"

Infinite recursion is also called a **stack overflow** -> your program has used up all the function stacks allowed to your program!

For the tower example, the base cases are when the height of the tower is 0 or 1

How to design a recursive function

Define your **recursion rule**

What should you do when your base cases don't apply?

Idea: Solve one step of the problem and then recurse on the remaining part of the problem

Tower Example:

Rule: To build a tower with height N , first build a tower with height $N-1$. Then add 1 block

Example: Recursive tower with height 3

To build a tower with height 3, first build a tower with height 2. Then add 1 block

How do we build a tower with height 2? Apply the rule again!

Example: Recursive tower with height 3

To build a tower with height 3, first build a tower with height 2. Then add 1 block

To build a tower with height 2, first build a tower with height 1. Then add 1 block

How do we build a tower with height 1? Apply the rule again!

Example: Recursive tower with height 3

To build a tower with height 3, first build a tower with height 2. Then add 1 block

To build a tower with height 2, first build a tower with height 1. Then add 1 block

To build a tower with height 1, place one block

base case! We know how to place a block!

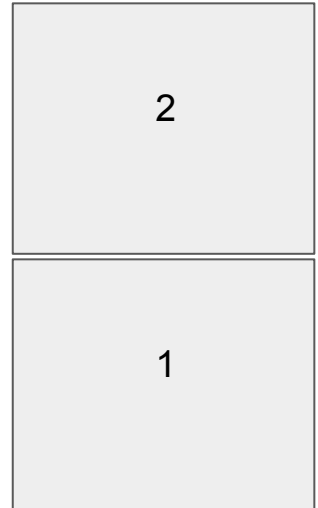


Example: Recursive tower with height 3

To build a tower with height 3, first build a tower with height 2. Then add 1 block

To build a tower with height 2, first build a tower with height 1. Then add 1 block

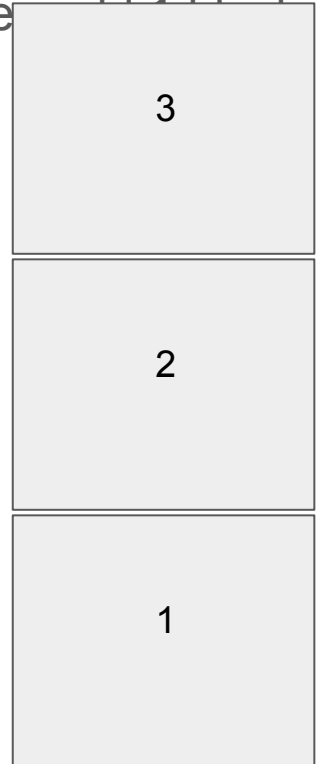
We have a tower of height 1, so add one block



Example: Recursive tower with height 3

To build a tower with height 3, first build a tower with height 2. The

We have a tower of height 2, so add one block



Example: printing a list

Iterative approach: For each element in the list, print it

Recursive approach:

- If the list is empty, return (base case)

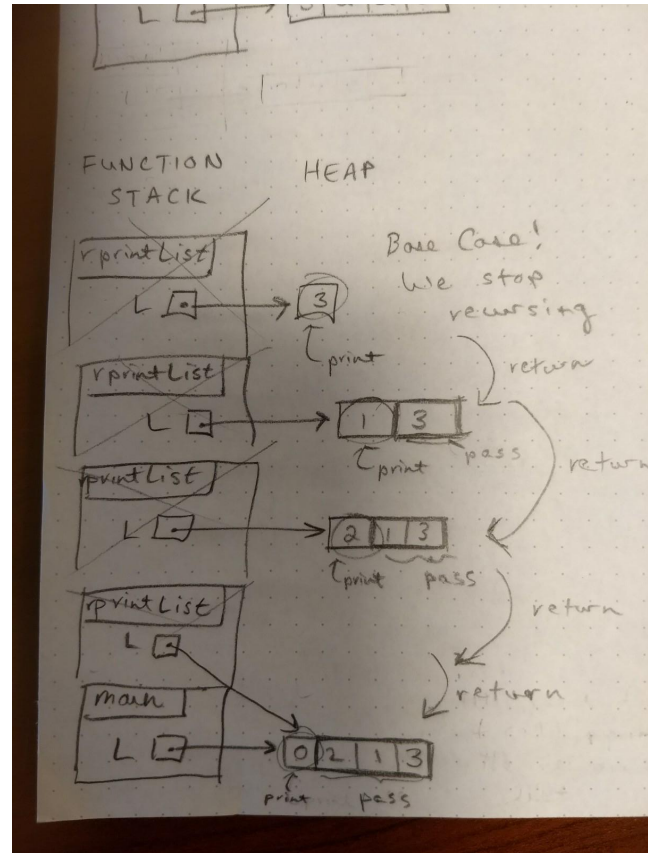
- If the list has one element, print it (base case)

- Otherwise, print the first element of the list and then print a list with size $N-1$ (recursion rule)

```

1 """
2 Write iterative and recursive functions to print a list
3 """
4
5 import random
6
7 def printList(L):
8     for i in range(len(L)):
9         print(L[i])
10
11 def rprintList(L):
12     print(L[0])
13     if len(L) > 1:
14         rprintList(L[1:])
15
16 def main():
17     L = list(range(5))
18     random.shuffle(L)
19     print(L)
20     print("-"*10)
21     printList(L)
22     print("-"*10)
23     rprintList(L)
24     print("-"*10)
25
26 main()

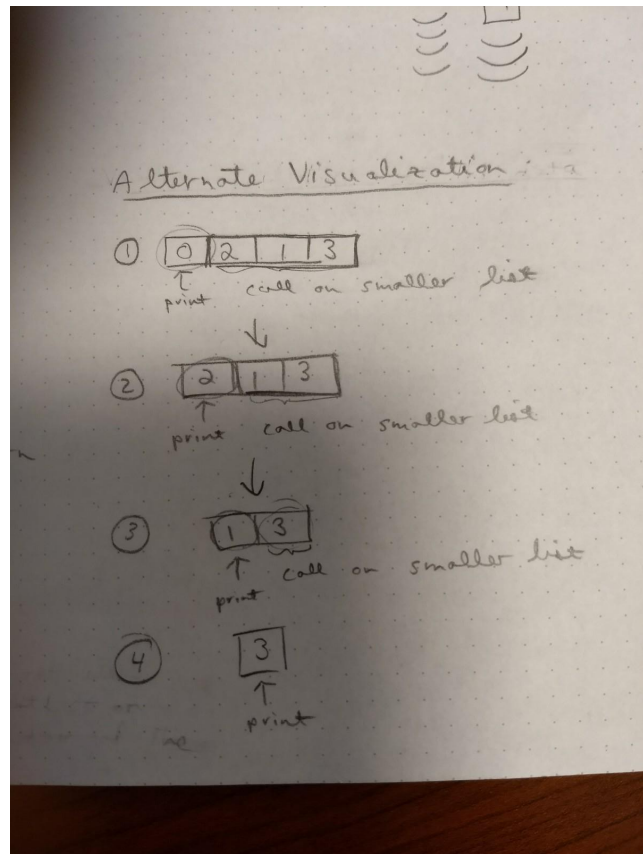
```



```

1 """
2 Write iterative and recursive functions to print a list
3 """
4
5 import random
6
7 def printList(L):
8     for i in range(len(L)):
9         print(L[i])
10
11 def rprintList(L):
12     print(L[0])
13     if len(L) > 1:
14         rprintList(L[1:])
15
16 def main():
17     L = list(range(5))
18     random.shuffle(L)
19     print(L)
20     print("-"*10)
21     printList(L)
22     print("-"*10)
23     rprintList(L)
24     print("-"*10)
25
26 main()

```



Example: multiplying elements in a list

Recursive approach:

If the list is empty, return 1 (base case)

If the list has one element, return it (base case)

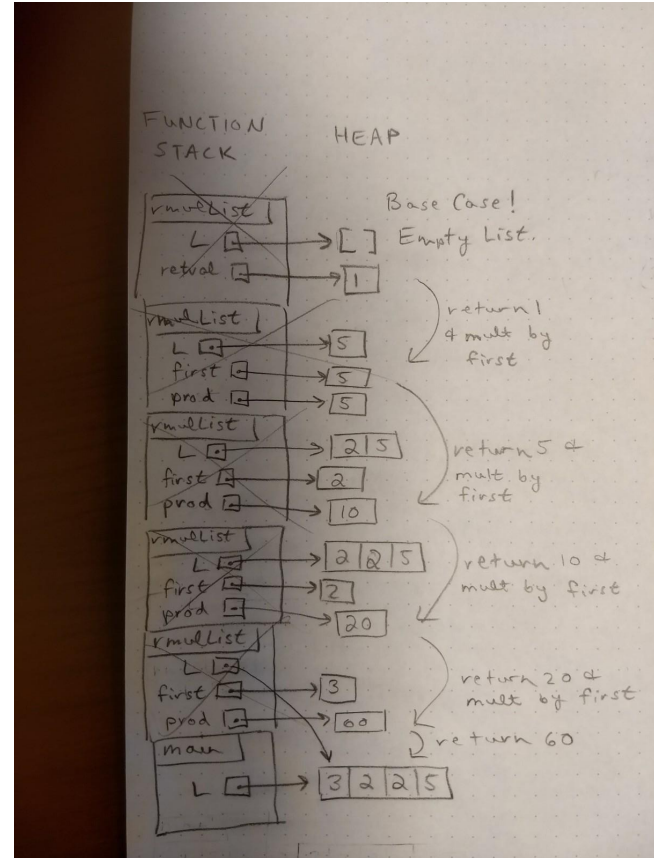
Otherwise, multiply the first element of the list with the product of the remaining elements (recursion rule)

```
1 """
2 Write iterative and recursive functions to multiply the items in a list
3 """
4
5 import random
6
7 def mullist(L):
8     prod = 1
9     for i in range(len(L)):
10         prod *= L[i]
11     return prod
12
13 def rmullist(L):
14     if len(L) == 0:
15         return 1
16     return L[0] * rmullist(L[1:])
17
18 def main():
19     L = [3, 5, 1, 2, 4]
20     assert(mullist(L) == 120)
21     assert(rmullist(L) == 120)
22     assert(rmullist([3]) == 3)
23     assert(rmullist([]) == 1)
24
25 main()
26
```

```

1  """
2  Write iterative and recursive functions to multiply the items in a list
3  """
4
5  import random
6
7  def mulList(L):
8      prod = 1
9      for i in range(len(L)):
10         prod *= L[i]
11     return prod
12
13  def rmullist(L):
14     if len(L) == 0:
15         return 1
16     first = L[0]
17     prod = first * rmullist(L[1:])
18     return prod
19
20  def main():
21     L = [3, 5, 1, 2, 4]
22     assert(mulList(L) == 120)
23     assert(rmullist(L) == 120)
24     assert(rmullist([3]) == 3)
25     assert(rmullist([]) == 1)
26
27  main()

```



```

1 """
2 Write iterative and recursive functions to multiply the items in a list
3 """
4
5 import random
6
7 def mulList(L):
8     prod = 1
9     for i in range(len(L)):
10        prod *= L[i]
11    return prod
12
13 def rmulList(L):
14     if len(L) == 0:
15        return 1
16    first = L[0]
17    prod = first * rmulList(L[1:])
18    return prod
19
20 def main():
21    L = [3, 5, 1, 2, 4]
22    assert(mulList(L) == 120)
23    assert(rmulList(L) == 120)
24    assert(rmulList([3]) == 3)
25    assert(rmulList([]) == 1)
26
27 main()

```

$$\begin{aligned}
 \text{rmulList}([3, 2, 2, 5]) &= 3 * \text{rmulList}([2, 2, 5]) \\
 &= 3 * (2 * \text{rmulList}([2, 5])) \\
 &= 3 * (2 * (2 * \text{rmulList}([5]))) \\
 &= 3 * (2 * (2 * (5 * \text{rmulList}([])))) \\
 &= 3 * (2 * (2 * (5 * 1))) \\
 &= 3 * (2 * (2 * 5)) \\
 &= 3 * (2 * 10) \\
 &= 3 * 20 \\
 &= 60
 \end{aligned}$$

3 by first
 2 by first
 2 by first
 5 by first
 10 by first
 20 by first
 60

Example: Sum the numbers 1 to n

Recursive approach:

If n is 0, return 0 (base case)

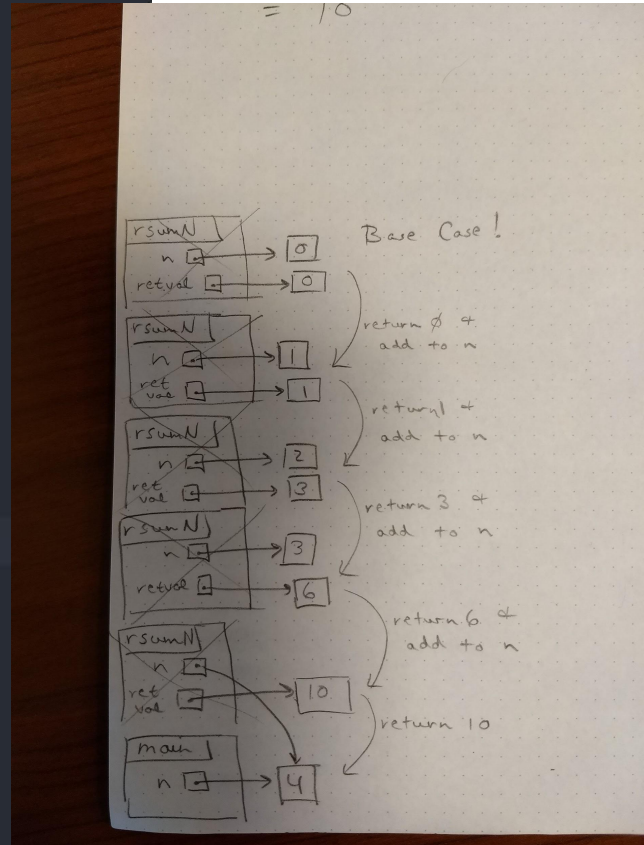
If n is 1, return 1 (base case)

Otherwise, multiply the first element of the list with the product of the remaining elements (recursion rule)

```

1  """
2  Given n, compute the sum from 0 to n (inclusive)
3  """
4
5  def sumN(n):
6      total = 0
7      for i in range(n+1):
8          total += i
9      return total
10
11 def rsumN(n):
12     if n == 0:
13         return 0
14     return n + rsumN(n-1)
15
16 def main():
17     print(rsumN(4))
18     #print(sumN(10))
19     #print(rsumN(10))
20
21 main()

```



```

1  """
2  Given n, compute the sum from 0 to n (inclusive)
3  """
4
5  def sumN(n):
6      total = 0
7      for i in range(n+1):
8          total += i
9      return total
10
11 def rsumN(n):
12     if n == 0:
13         return 0
14     return n + rsumN(n-1)
15
16 def main():
17     print(rsumN(4))
18     #print(sumN(10))
19     #print(rsumN(10))
20
21 main()

```

$$\begin{aligned}
 rsumN(4) &= 4 + rsumN(3) \\
 &= 4 + [3 + rsumN(2)] \\
 &= 4 + [3 + [2 + rsumN(1)]] \\
 &= 4 + [3 + [2 + [1 + rsumN(0)]]] \\
 &= 4 + [3 + [2 + [1 + 0]]] \\
 &= 4 + [3 + [2 + 1]] \\
 &= 4 + [3 + 3] \\
 &= 4 + 6 \\
 &= 10
 \end{aligned}$$

Example: Print hello n times

n is a counter in this example

Recursive approach:

If n is 0, do nothing (base case)

Otherwise, print “hello” and repeat n-1 more times (recursion rule)

```
1 """
2 Print "hello" n times
3 """
4
5 def hello(n):
6     for i in range(n):
7         print("hello")
8
9 def rhello(n):
10     if n > 0:
11         print("hello")
12         rhello(n-1)
13
14
15 def main():
16     hello(3)
17     print("-"*10)
18     rhello(3)
19
20 main()
21
```

If $n = 3$, can you draw the function stack?

Can you find the error?

What is the output of this program?

```
1 """
2 Write iterative and recursive functions
3 """
4
5 import random
6
7 def printList(L):
8     for i in range(len(L)):
9         print(L[i])
10
11 def rprintList(L):
12     print(L[0])
13     if len(L) > 1:
14         rprintList(L)
15
16 def main():
17     L = list(range(5))
18     random.shuffle(L)
19     print(L)
20     print("-"*10)
21     printList(L)
22     print("-"*10)
23     rprintList(L)
24     print("-"*10)
25
26 main()
```

Infinite recursion

What happened?

How to fix?

```
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
Traceback (most recent call last):
  File "printList.py", line 26, in <module>
    main()
  File "printList.py", line 23, in main
    rprintList(L)
  File "printList.py", line 14, in rprintList
    rprintList(L)
  File "printList.py", line 14, in rprintList
    rprintList(L)
  File "printList.py", line 14, in rprintList
    rprintList(L)
  File "printList.py", line 14, in rprintList
    rprintList(L)
  [Previous line repeated 991 more times]
  File "printList.py", line 12, in rprintList
    print(L[0])
RecursionError: maximum recursion depth exceeded while calling a Python object
almond[w13]$
```

Can you find the error?

```
1 """
2 Write iterative and recursive functions
3 """
4
5 import random
6
7 def printList(L):
8     for i in range(len(L)):
9         print(L[i])
10
11 def rprintList(L):
12     print(L[0])
13     if len(L) > 1:
14         rprintList(L)
15
16 def main():
17     L = list(range(5))
18     random.shuffle(L)
19     print(L)
20     print("-"*10)
21     printList(L)
22     print("-"*10)
23     rprintList(L)
24     print("-"*10)
25
26 main()
```

We never reach the base case because we pass the whole list!

We should pass L[1:]