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: The 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!

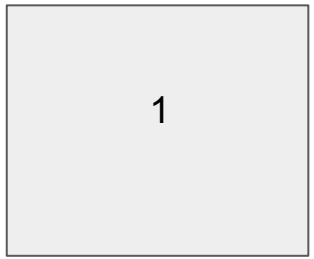
Example: Recursive tower with height 3

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!

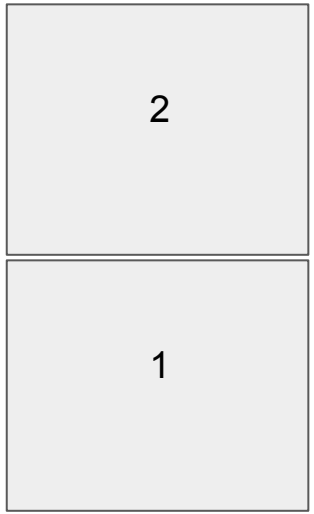


Example: Recursive tower with height 3

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

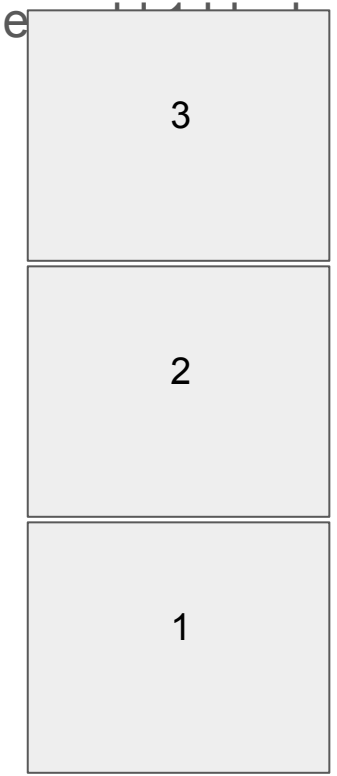
We have a tower of height 1, so add one block



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



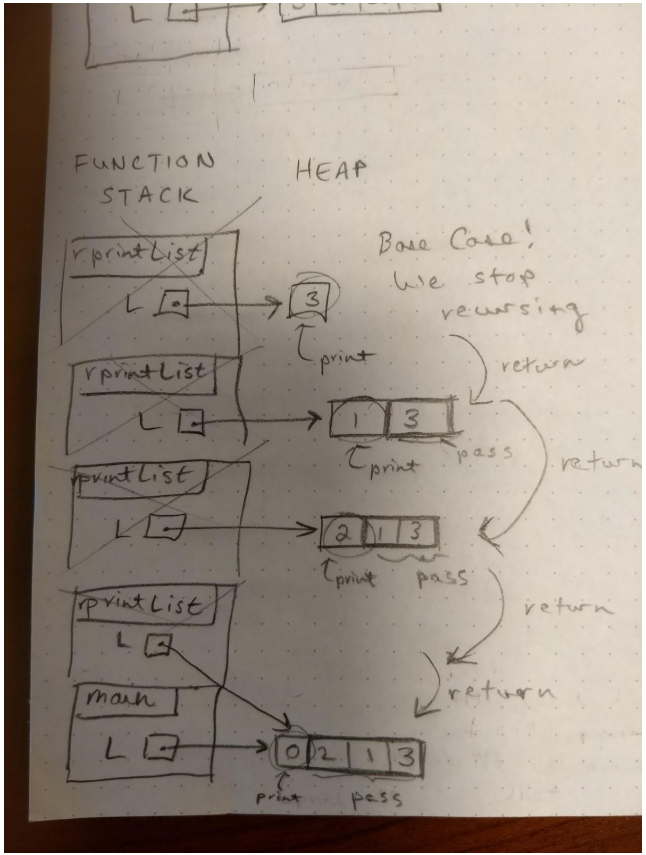
Example: printing a list

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

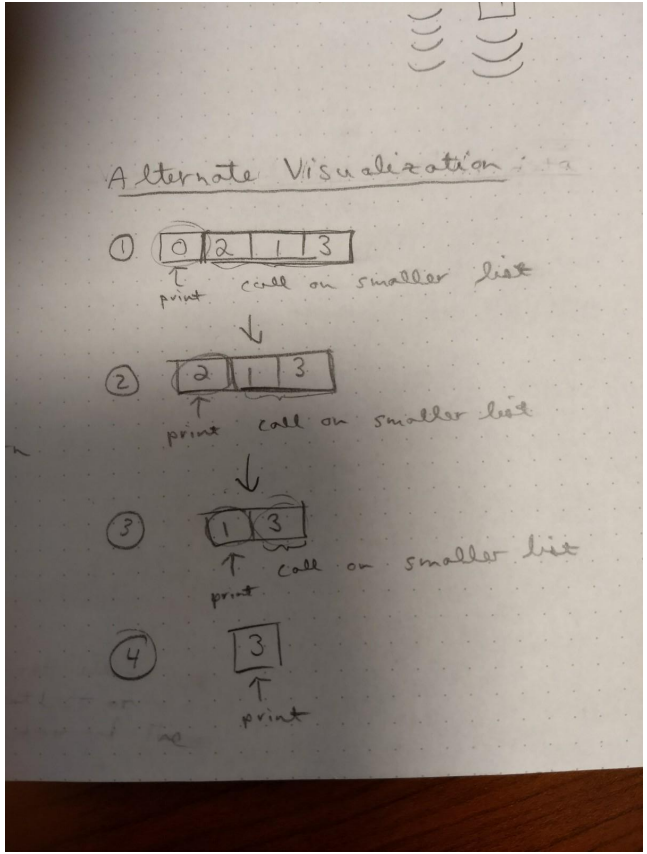
Recursive approach:

- 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)

```python
"""
Write iterative and recursive functions to print a list
"""

import random

def printList(L):
    for i in range(len(L)):
        print(L[i])

def rprintList(L):
    print(L[0])
    if len(L) > 1:
        rprintList(L[1:])

def main():
    L = list(range(5))
    random.shuffle(L)
    print(L)
    print("-"*10)
    printList(L)
    print("-"*10)
    rprintList(L)
    print("-"*10)

main()
```



```python
"""
Write iterative and recursive functions to print a list
"""

import random

def printList(L):
    for i in range(len(L)):
       print(L[i])

def rprintList(L):
    print(L[0])
    if len(L) > 1:
        rprintList(L[1:])

def main():
    L = list(range(5))
    random.shuffle(L)
    print(L)
    print("-"*10)
    printList(L)
    print("-"*10)
    rprintList(L)
    print("-"*10)

main()
```



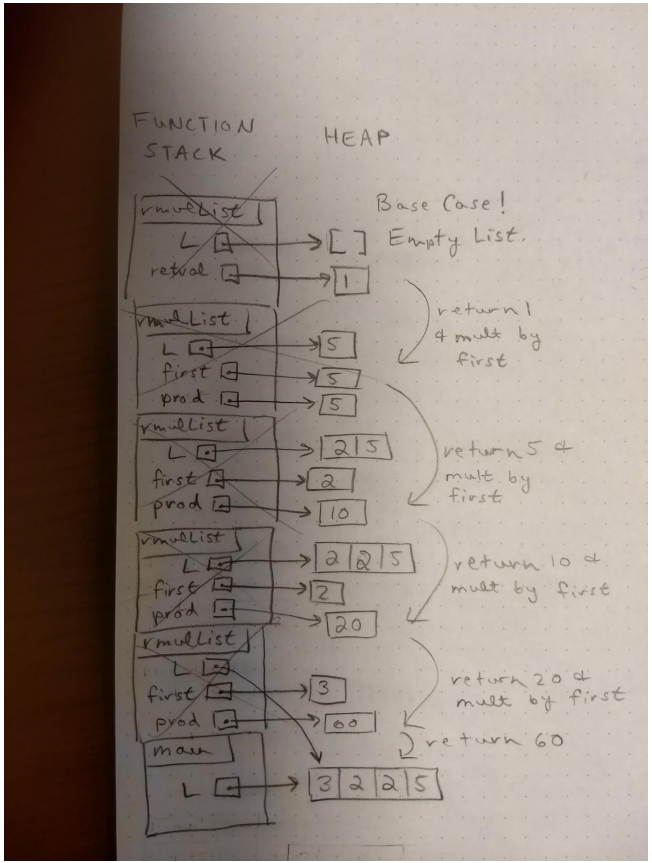
Example: multiplying elements in a list

Recursive approach:

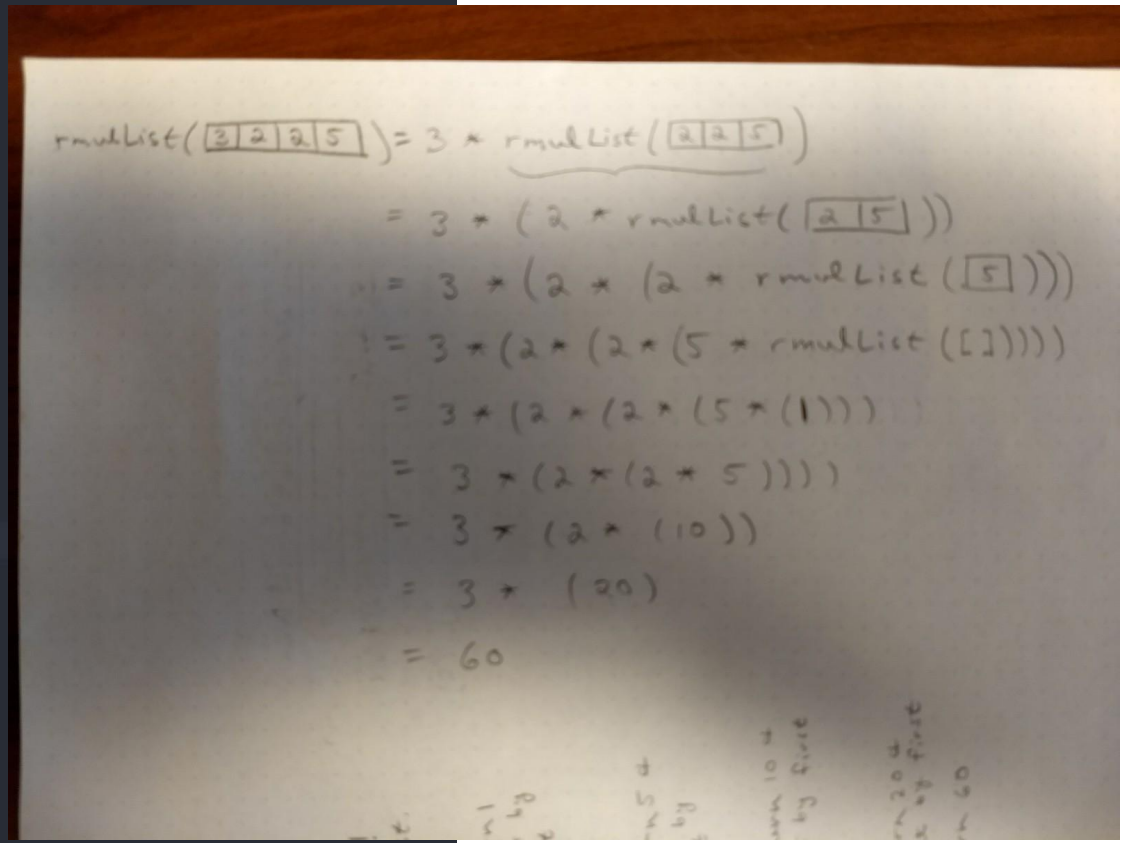
- If the list is empty, return 1 (base case)
- If the list has one element, return it (base case)
- Otherwise, multiply the first element of the list with the product of the remaining elements (recursion rule)

```
"""
Write iterative and recursive functions to multiply the items in a list
"""

import random

def mulList(L):
    prod = 1
    for i in range(len(L)):
       prod *= L[i]
    return prod

def rmulList(L):
    if len(L) == 0:
        return 1
    return L[0] * rmulList(L[1:])

def main():
    L = [3, 5, 1, 2, 4]
    assert(mulList(L) == 120)
    assert(rmulList(L) == 120)
    assert(rmulList([3]) == 3)
    assert(rmulList([]) == 1)

main()

```

```python
"""
Write iterative and recursive functions to multiply the items in a list
"""

import random

def mulList(L):
    prod = 1
    for i in range(len(L)):
        prod *= L[i]
    return prod

def rmulList(L):
    if len(L) == 0:
        return 1
    first = L[0]
    prod = first * rmulList(L[1:])
    return prod

def main():
    L = [3, 5, 1, 2, 4]
    assert(mulList(L) == 120)
    assert(rmulList(L) == 120)
    assert(rmulList([3]) == 3)
    assert(rmulList([]) == 1)

main()
```



```python
"""
Write iterative and recursive functions to multiply the items in a list
"""

import random

def mulList(L):
    prod = 1
    for i in range(len(L)):
        prod *= L[i]
    return prod

def rmulList(L):
    if len(L) == 0:
        return 1
    first = L[0]
    prod = first * rmulList(L[1:])
    return prod

def main():
    L = [3, 5, 1, 2, 4]
    assert(mulList(L) == 120)
    assert(rmulList(L) == 120)
    assert(rmulList([3]) == 3)
    assert(rmulList([]) == 1)

main()
```



Example: Sum the numbers 1 to n

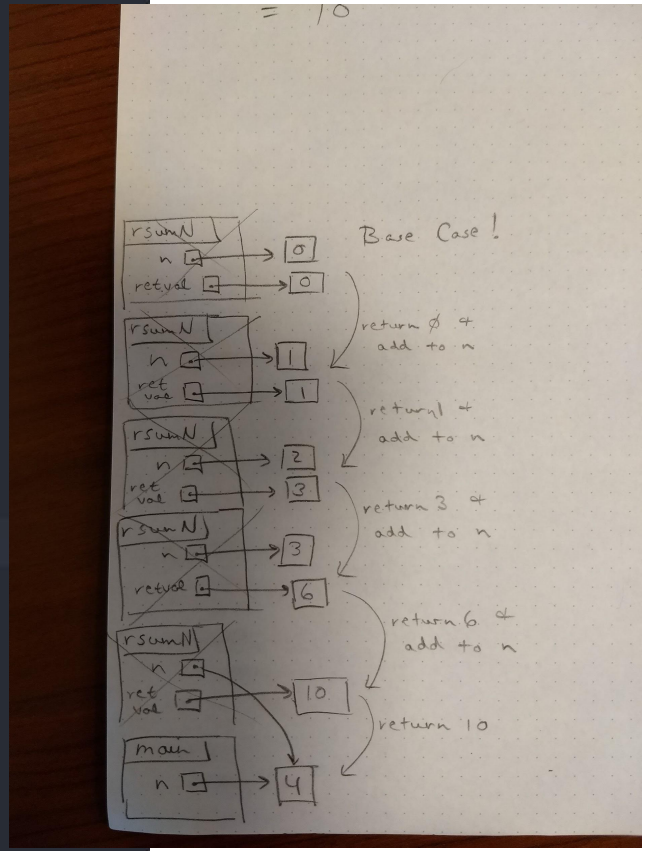
Recursive approach:

If n is 0, return 0 (base case)

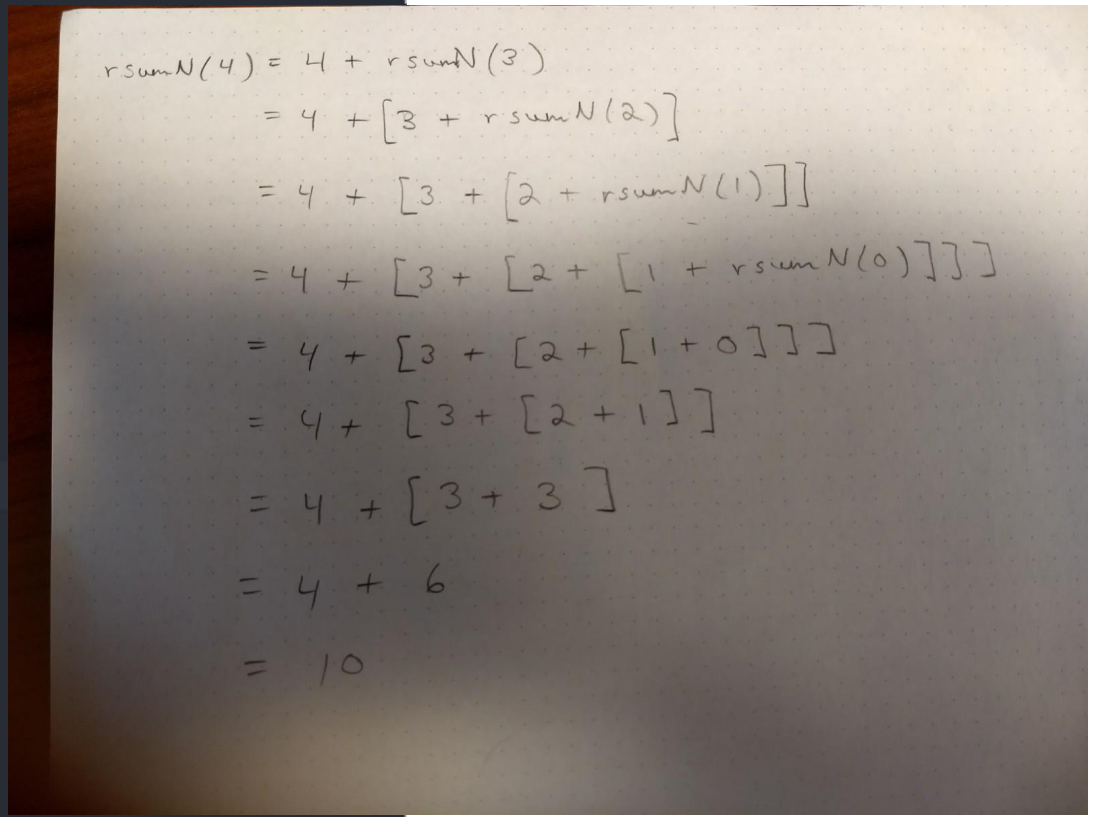
If n is 1, return 1 (base case)

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

```python
"""
Given n, compute the sum from 0 to n (inclusive)
"""

def sumN(n):
    total = 0
    for i in range(n+1):
        total += i
    return total

def rsumN(n):
    if n == 0:
        return 0
    return n + rsumN(n-1)

def main():
    print(rsumN(4))
    #print(sumN(10))
    #print(rsumN(10))

main()
```



```python
"""
Given n, compute the sum from 0 to n (inclusive)
"""

def sumN(n):
    total = 0
    for i in range(n+1):
        total += i
    return total

def rsumN(n):
    if n == 0:
        return 0
    return n + rsumN(n-1)

def main():
    print(rsumN(4))
    #print(sumN(10))
    #print(rsumN(10))

main()
```



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)

```
"""
Print "hello" n times
"""

def hello(n):
    for i in range(n):
        print("hello")

def rhello(n):
    if n > 0:
        print("hello")
        rhello(n-1)


def main():
    hello(3)
    print("-"*10)
    rhello(3)

main()

```

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

Can you find the error?

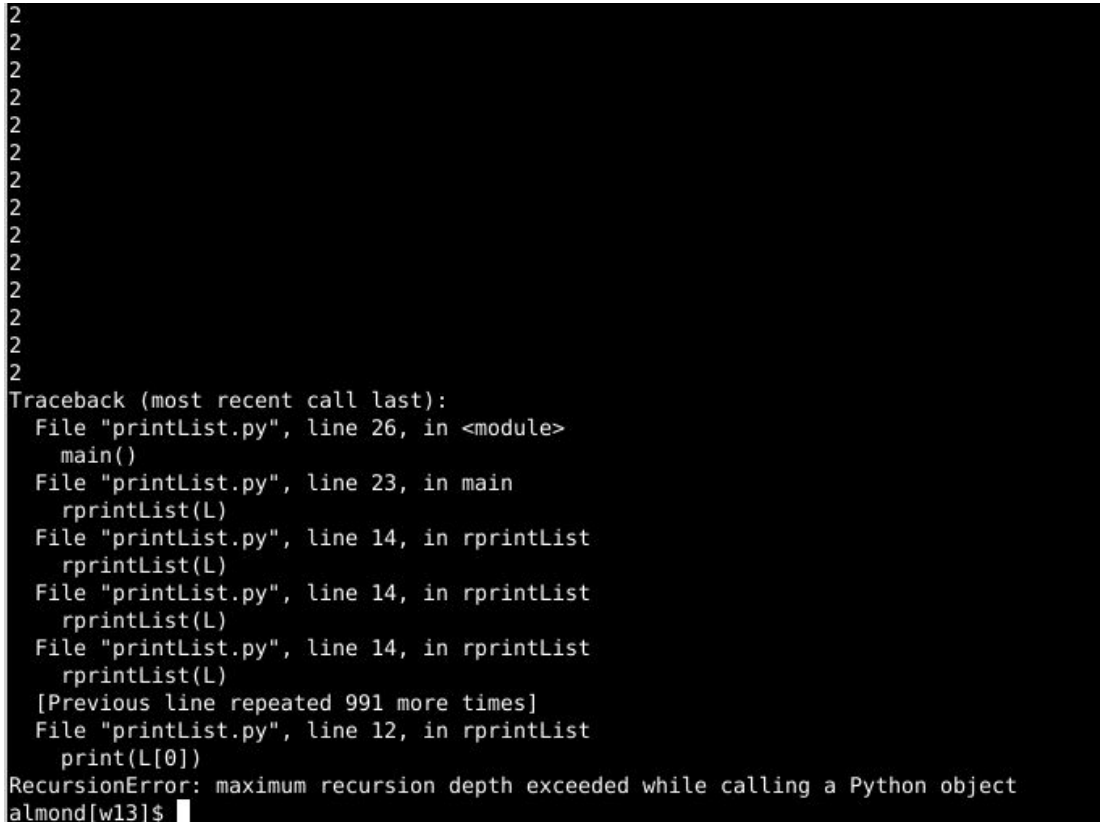
What is the output of this program?

```
"""
Write iterative and recursive functions
"""

import random

def printList(L):
    for i in range(len(L)):
       print(L[i])

def rprintList(L):
    print(L[0])
    if len(L) > 1:
        rprintList(L)

def main():
    L = list(range(5))
    random.shuffle(L)
    print(L)
    print("-"*10)
    printList(L)
    print("-"*10)
    rprintList(L)
    print("-"*10)

main()
```

Infinite recursion

What happened?

How to fix?



Can you find the error?

```
"""
Write iterative and recursive functions
"""

import random

def printList(L):
    for i in range(len(L)):
        print(L[i])

def rprintList(L):
    print(L[0])
    if len(L) > 1:
        rprintList(L)

def main():
    L = list(range(5))
    random.shuffle(L)
    print(L)
    print("-"*10)
    printList(L)
    print("-"*10)
    rprintList(L)
    print("-"*10)

main()
```

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

We should pass L[1:]