Recursion

CS 1.3 - Core Data Structures

Recursion

Calculating the Sum of a List of Numbers

MAKE SCHOOL

Using Python, how would you compute the sum of a list of numbers?

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Calculating the Sum of a List of Number

Using Python, how would you compute the sum of a list of numbers?

```
def list_sum(num_list):
    sum = 0
    for i in num_list:
        sum += i
    return sum
```

```
print(list_sum([1,3,5,7,9]))
```

Calculating the Sum of a List of Number

Using Python, how would you compute the sum of a list of numbers?

BUT…

You cannot use while and for loops.

Ideas?

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(16))))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+16)))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

$$total = (1+(3+(21)))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+16)))$$

$$total = (1+(3+21))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+16)))$$

$$total = (1+(24))$$

Calculating the Sum of a List of Number

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+16)))$$

$$total = (1+(3+21))$$

$$total = (1+24)$$

Calculating the Sum of a List of Number

MAKE SCHOOL

$$(1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+(7+9))))$$

$$total = (1+(3+(5+16)))$$

$$total = (1+(3+21))$$

$$total = (1+24)$$

$$total = 25$$

Calculating the Sum of a List of Number

$$listSum(numList) = first(numList) + listSum(rest(numList))$$

(not code, mathematical equation)

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + sum([9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + sum([9])

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + sum([9])

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + 9

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + 9

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + 16

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + 16

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + 21

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + 21

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 24

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 24

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + 24

sum([3, 5, 7, 9]) = 24

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + 24

sum([3, 5, 7, 9]) = 24

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 25

sum([3, 5, 7, 9]) = 24

sum([5, 7, 9]) = 21

sum([7, 9]) = 16

sum([9]) = 9

What is recursion?

In Computer Science, **recursion** is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

Recursion usually involves a function calling itself.

Calculating the Sum of a List of Number

MAKE SCHOOL

```
def list_sum(lst):

    sum = 0
        for i in lst:
                sum = sum + i
        return sum
```

Iterative

```
def list_sum(lst):
  if len(lst) == 1:
    return lst[0]
  else:
    return lst[0] + list_sum(lst[1:])
```

Recursive

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Three Laws of Recursions

1. A recursive algorithm must have a **base case**.
2. A recursive algorithm must change its state and move toward the base case.
3. A recursive algorithm must call itself, recursively.

Three Laws of Recursion

MAKE SCHOOL

1. A recursive algorithm must have a **base case**.

2. A recursive algorithm must change its state and move toward the base case.

3. A recursive algorithm must call itself, recursively.

```
def list_sum(lst):
  if len(lst) == 1:
    return lst[0]
  else:
    return lst[0] + list_sum(lst[1:])
```

Recursive

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + 9

sum([9]) = 9

Three Laws of Recursion

1. A recursive algorithm must have a **base case**.
2. A recursive algorithm must change its state and move toward the base case.
3. A recursive algorithm must call itself, recursively.

```
def list_sum(lst):
  if len(lst) == 1:
    return lst[0]
  else:
    return lst[0] + list_sum(lst[1:])
```

Recursive

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + sum([9])

Calculating the Sum of a List of Number

sum([1,3,5,7,9]) = 1 + sum([3, 5, 7, 9])

sum([3, 5, 7, 9]) = 3 + sum([5, 7, 9])

sum([5, 7, 9]) = 5 + sum([7, 9])

sum([7, 9]) = 7 + sum([9])

sum([9]) = 9

Three Laws of Recursion

1. A recursive algorithm must have a **base case**.

2. A recursive algorithm must change its state and move toward the base case.

3. A recursive algorithm must call itself, recursively.

```
def list_sum(lst):
  if len(lst) == 1:
    return lst[0]
  else:
    return lst[0] + list_sum(lst[1:])
```

Recursive

How can we make the Binary Search Algorithm recursive?

Apply the 3 Laws.

- What is the base case?
- What state is being changed?
- What would we use as arguments when we call the function again?

Let’s code it!

Students browse: repl.it/@MakeSchool/BinarySearch?lite=true

Pear Deck Interactive Slide
Do not remove this bar

Factorial

A factorial is when a given number is multiplied by each number less than it.

Use the "!" symbol to represent a factorial

i.e. 4! = 4 * 3 * 2 * 1 = 24

What is 5!

How could we write this more generally?

Factorial

MAKE SCHOOL

n! = n * (n-1) *)n-2) * ... * 2 * 1

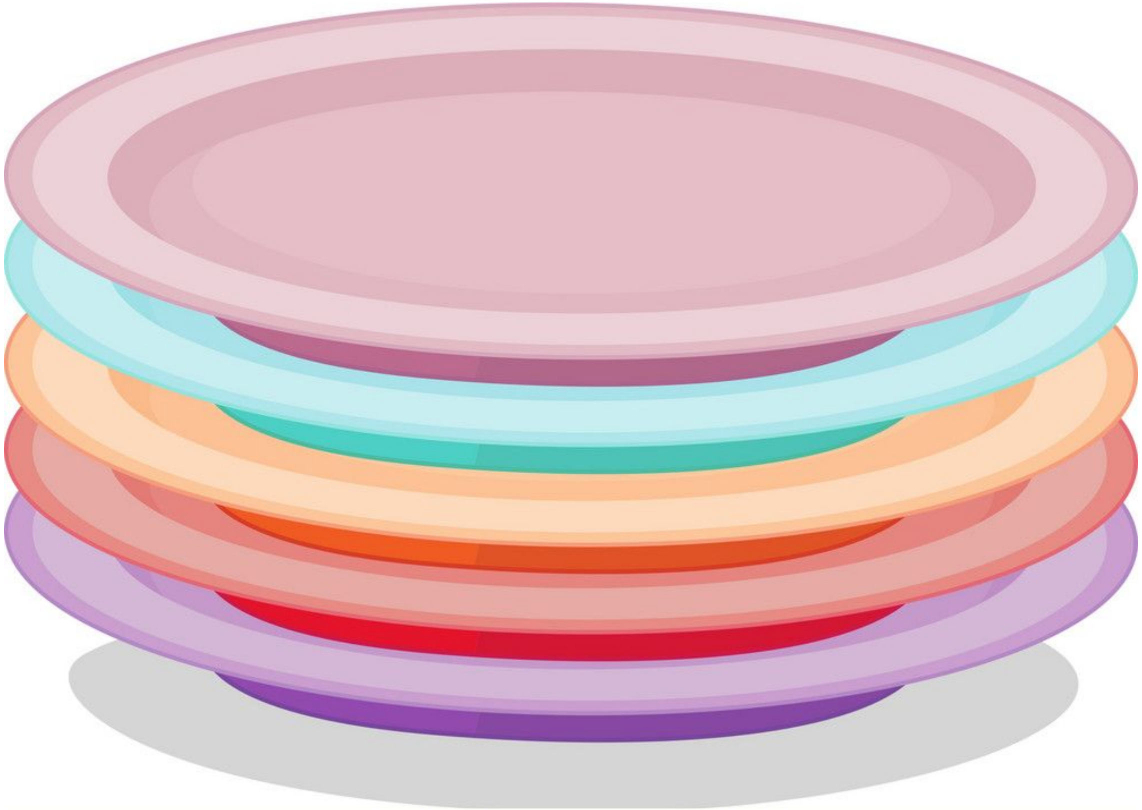
n! = n * (n-1)!

You can define factorials in terms of each other!

```python
def factorial(n):
    if n == 1:
        return 1
    else:
        f = factorial(n-1)
        return n*f
```

Base Case

Visualization: the call stack



Drawing a call stack

Code Trace Call Stack for factorial(3)

Call stack for recursive factorial, factorial(3)

~~factorial(1)~~ → 1 ret

~~factorial(2)~~ → f = 1 2 * 1 ret
n = 2

~~factorial(3)~~ → f = 2 ret
n = 3 2 * 3

Bottom of stack factorial(3) → 6

~~3~~ ~~2~~ 1

```python
def factorial(n):
    if n == 1:  x x ✓
        return 1
    else:
        f = factorial(n-1)
        return n*f
```

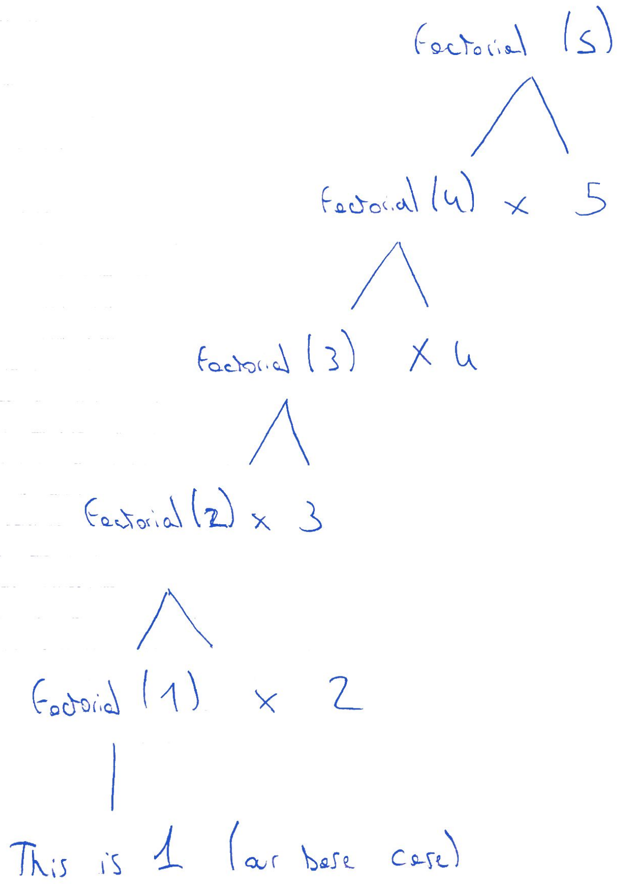
2 - 1
~~3 - 1~~

Draw the call stack for factorial(2)

MAKE SCHOOL

```
def factorial(n):
    if n == 1:
        return 1
    else:
        f = factorial(n-1)
        return n*f
```

Visualization: Recursive Trees



Draw the recursive tree for factorial(3)

MAKE SCHOOL

Recursion vs. Iteration

- You can rewrite every recursive algorithm as an iterative one
- Recursion isn’t always better, it often will use more memory than iteration
- In some cases, recursion may not be as efficient as iteration but it is much more elegant and readable

Tail Recursion: Optimization

MAKE SCHOOL

Tail recursion is the idea that we will write our function in such a way that the recursive call is the last operation we perform on a non base case

Why would this be a useful way to optimize?

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Module 6: Recursion

MAKE SCHOOL