Mathematical Foundations

Recursion

CS51 – Spring 2026

Programs so far

- Contain control flow and iteration.
- We will see a new way of solving problems using **recursion**.

Practice time

- The factorial of a non-negative integer n, denoted as $n! = 1 \times 2 \times 3 \times \cdots (n-1) \times n$
- For example, $5! = 1 \times 2 \times 3 \times 4 \times 5 \ = \ 120$.

- Write a function called factorial that takes a number as its single parameter and returns the factorial of that number.

Answer

```
def factorial_iterative(n):
    num = 1
    for i in range(2, n + 1):
        num *= i  # same as num = num * i
    return num
def factorial_iterative2(n):
    num = 1
    for i in range(n):
        num *= (i + 1)  # same as num = num * (i+1)
    return num
```

Recursion

- A recursive function is defined with respect to itself:
  - The function calls itself.
  - The recursive call should be on a "smaller" version of the problem.
- Can we write the factorial function recursively?
  - key idea: try and break down the problem into some computation, plus a smaller subproblem that looks similar.
    - $5! = 5 \times 4 \times 3 \times 2 \times 1$
    - $5! = 5 \times 4!$
  - How does this translate to Python code?

A first try at a recursive factorial function

```
def factorial(n):
    return n * factorial(n - 1)
```

- What happens if we call factorial(5)?

```
> 5*factorial(4)
  > 4*factorial(3)
    > 3*factorial(2)
       > 2*factorial(1)
        > 1*factorial(0)
          > 0*factorial(-1)
           > ....
```

- At some point we need to stop. this is called the **base case** for recursion. When?
  - When n == 1.

A recursive factorial function

```
def factorial(n):
    # check the base case
    if n == 1:
        return 1
    else:
        # the recursive case
        return n * factorial(n - 1)
```

A recursive factorial function

```
def factorial(n):
    # check the base case
    if n == 1:
        return 1
    else:
        # the recursive case
        return n * factorial(n - 1)
```

What happens in the call stack when we call factorial(5)?

A recursive factorial function

```
def factorial(n):
    # check the base case
    if n == 1:
        return 1
    else:
        # the recursive case
        return n * factorial(n - 1)

> factorial(5)
```

factorial(5)

A recursive factorial function

```
def factorial(n):
    # check the base case
    if n == 1:
        return 1
    else:
        # the recursive case
        return n * factorial(n - 1)
```

- ➢ factorial(5)
    - ➢ 5*factorial(4)

n=5

factorial(5)
    return 5 * factorial(4)

A recursive factorial function

```
def factorial(n):
    # check the base case
    if n == 1:
        return 1
    else:
        # the recursive case
        return n * factorial(n - 1)
```

- factorial(5)
    - 5*factorial(4)
        - 4*factorial(3)

n=4

factorial(4)
return 4 * factorial(3)

n=5

factorial(5)
return 5 * factorial(4)

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- factorial(5)
  - 5*factorial(4)
    - 4*factorial(3)
      - 3*factorial(2)

n=3

factorial(3)
    return 3 * factorial(2)

n=4

factorial(4)
    return 4 * factorial(3)

n=5

factorial(5)
    return 5 * factorial(4)

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- factorial(5)
  - 5*factorial(4)
    - 4*factorial(3)
      - 3*factorial(2)
        - 2*factorial(1)

n=2

factorial(2)
    return 2 * factorial(1)

n=3

factorial(3)
    return 3 * factorial(2)

n=4

factorial(4)
    return 4 * factorial(3)

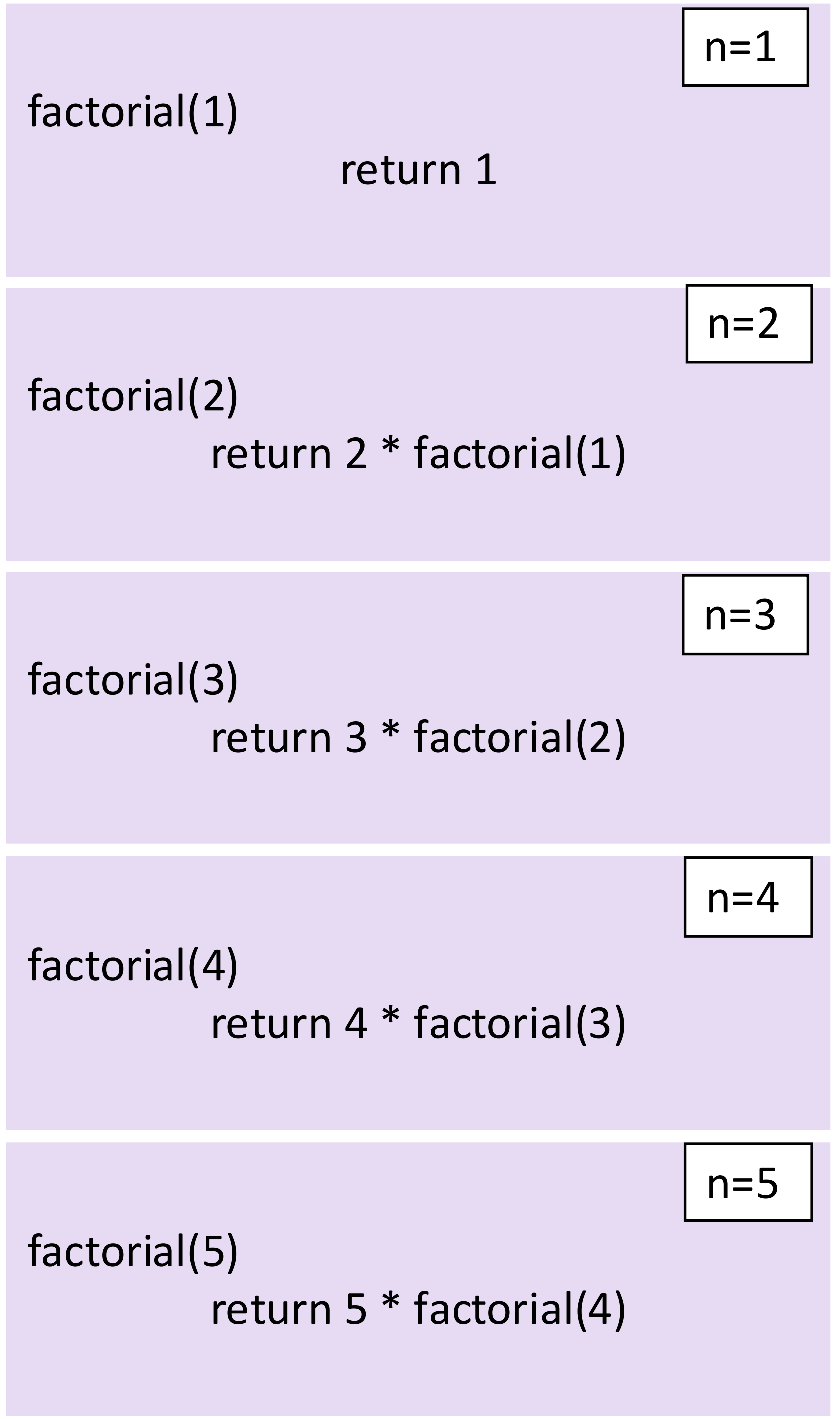
n=5

factorial(5)
    return 5 * factorial(4)

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- factorial(5)
  - 5*factorial(4)
    - 4*factorial(3)
      - 3*factorial(2)
        - 2*factorial(1)
          - 1



```
n=1
factorial(1)
        return 1

n=2
factorial(2)
        return 2 * factorial(1)

n=3
factorial(3)
        return 3 * factorial(2)

n=4
factorial(4)
        return 4 * factorial(3)

n=5
factorial(5)
        return 5 * factorial(4)
```

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- factorial(5)
  - 5*factorial(4)
    - 4*factorial(3)
      - 3*factorial(2)
        - 2*factorial(1)
          - 1
        - 2*1 = 2

n=2

```
factorial(2)
        return 2 * 1 = 2
```

n=3

```
factorial(3)
      return 3 * factorial(2)
```

n=4

```
factorial(4)
      return 4 * factorial(3)
```

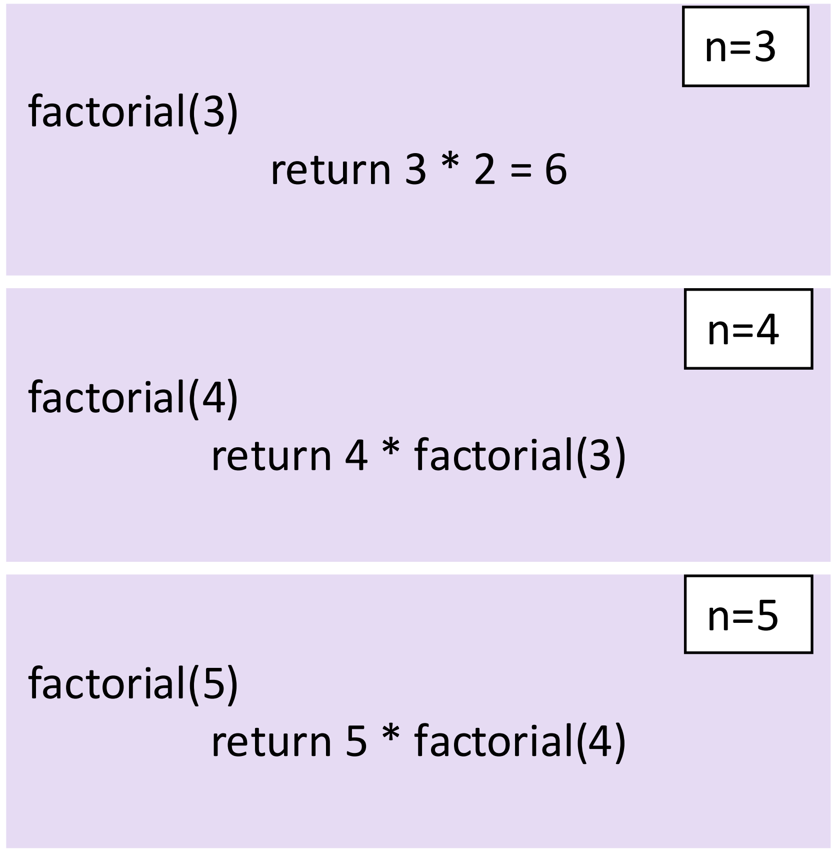
n=5

```
factorial(5)
      return 5 * factorial(4)
```

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- ➢ factorial(5)
  - ➢ 5*factorial(4)
    - ➢ 4*factorial(3)
      - ➢ 3*factorial(2)
        - ➢ 2*factorial(1)
          - ➢ 1
        - ➢ 2*1 = 2
      - ➢ 3*2 = 6



A recursive factorial function

```
def factorial(n):
    # check the base case
    if n == 1:
        return 1
    else:
        # the recursive case
        return n * factorial(n - 1)
```

- factorial(5)
    - 5*factorial(4)
        - 4*factorial(3)
            - 3*factorial(2)
                - 2*factorial(1)
                    - 1
                - 2*1 = 2
            - 3*2 = 6
        - 4*6 = 24

n=4

factorial(4)
return 4 * 6 = 24

n=5

factorial(5)
return 5 * factorial(4)

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- factorial(5)
  - 5*factorial(4)
    - 4*factorial(3)
      - 3*factorial(2)
        - 2*factorial(1)
          - 1
        - 2*1 = 2
      - 3*2 = 6
    - 4*6 = 24
  - 5*24 = 120

n=5

factorial(5)

return 5 * 24 = 120

A recursive factorial function

```
def factorial(n):
  # check the base case
  if n == 1:
    return 1
  else:
    # the recursive case
    return n * factorial(n - 1)
```

- factorial(5)
  - 5*factorial(4)
    - 4*factorial(3)
      - 3*factorial(2)
        - 2*factorial(1)
          - 1
        - 2*1 = 2
      - 3*2 = 6
    - 4*6 = 24
  - 5*24 = 120
- 120

Writing recursive functions

1. Define what the function the name and parameters of the function are.
2. Define the recursive case
3. Define the base case
4. Put it all together

Recursion is similar to induction

- Proof by principle of induction:
  - 1. show something works for $P(0)$ (base case).
  - 2. create an induction hypothesis $P(n)$ for some arbitrary n
  - 3. show it will work for $P(n+1)$ (recursive case)
  - 4. therefore, it must work for every n.

Practice time

- Write a recursive function called `rec_sum` that takes a positive number as a parameter and calculates the sum of the numbers from 0 up to and including that provided number.
- Remember the steps:

1. Define what the function the name and parameters of the function are.
2. Define the recursive case
3. Define the base case
4. Put it all together

Answer

1. Define what the function the name and parameters of the function are.
    - def rec_sum(n)
2. Define the recursive case
    - rec_sum(n) = n + rec_sum(n-1)
    - i.e. the sum of the numbers 1 through n, is n plus the sum of the numbers 1 through n-1
3. Define the base case
    - in each case, the number is getting smaller. What’s the smallest number we would ever want to have the sum of?
        - 0. What’s the answer when it's 0? 0!
4. Put it all together

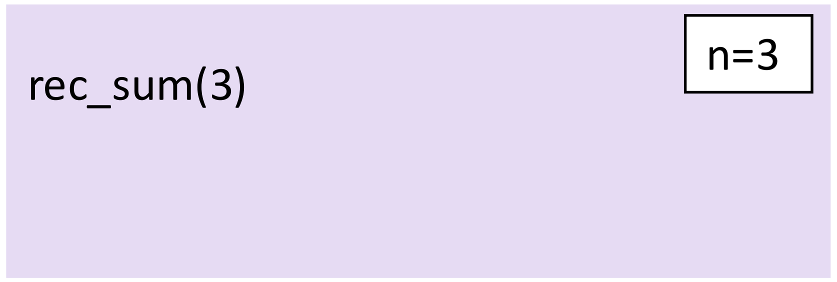
Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

- rec_sum(3)



Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

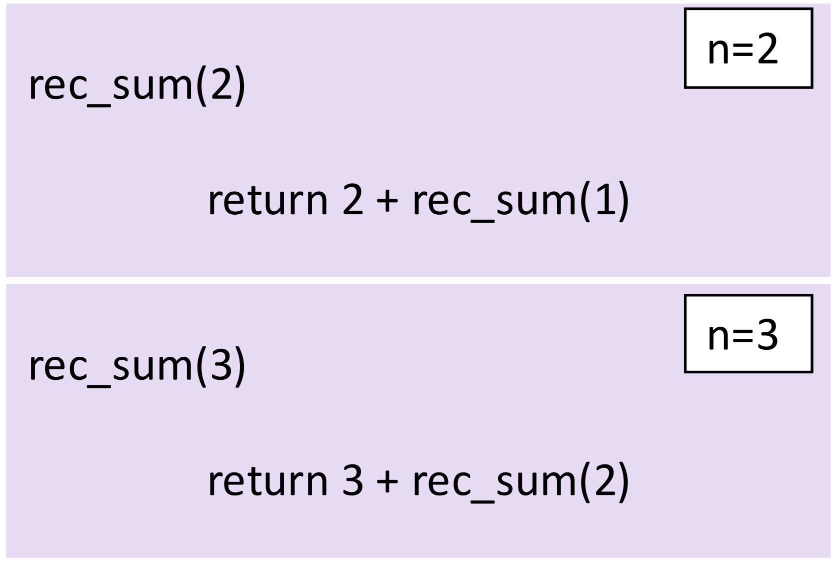
- rec_sum(3)
  - 3 + rec_sum(2)

```
rec_sum(3)                          n=3

        return 3 + rec_sum(2)
```

Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

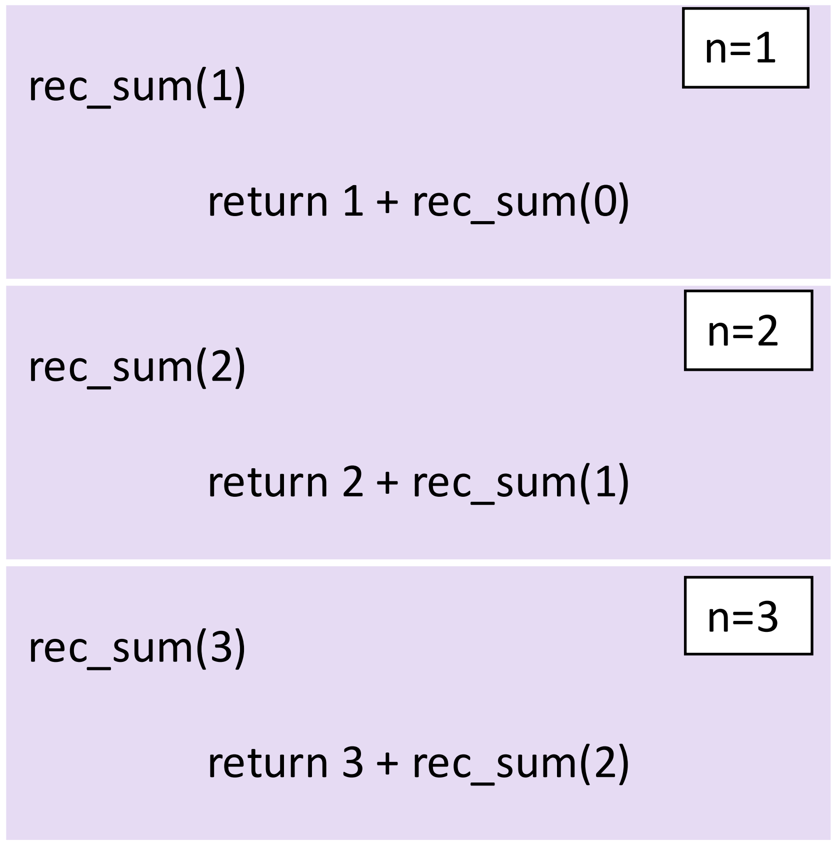
- rec_sum(3)
  - 3 + rec_sum(2)
    - 2 + rec_sum(1)



Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

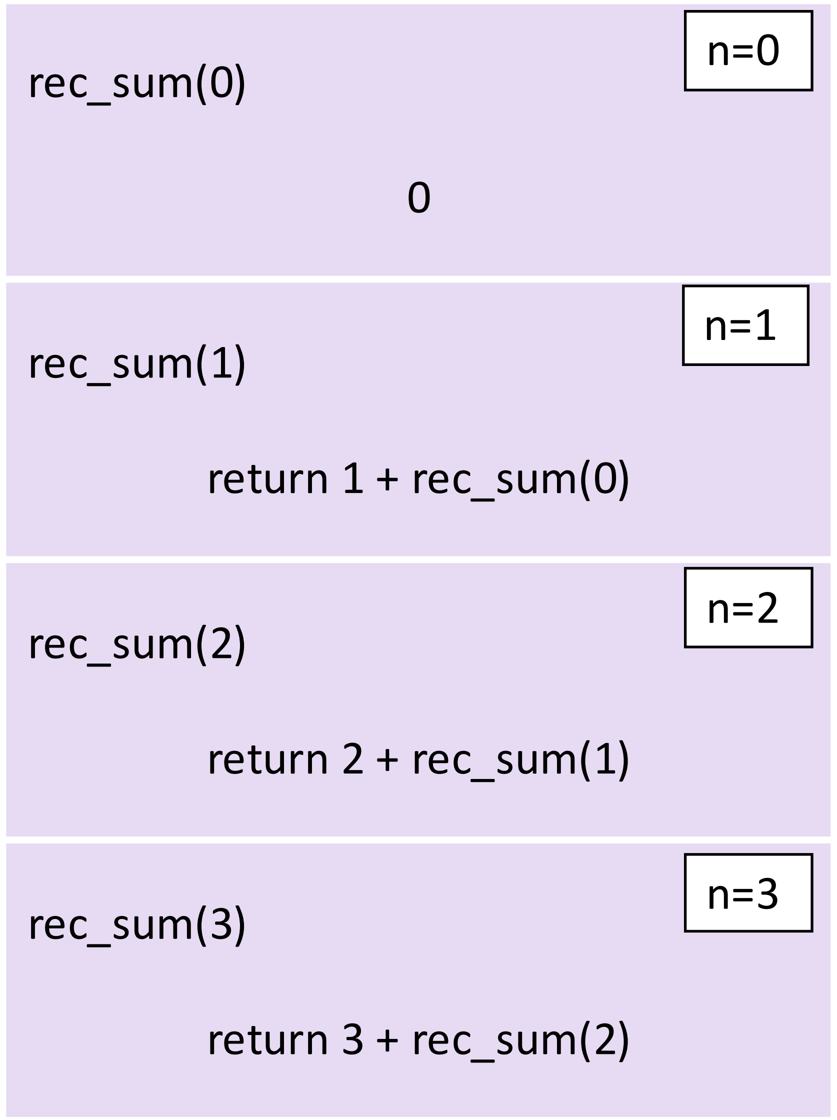
- rec_sum(3)
    - 3 + rec_sum(2)
        - 2 + rec_sum(1)
            - 1 + rec_sum(0)



Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

- rec_sum(3)
    - 3 + rec_sum(2)
        - 2 + rec_sum(1)
            - 1 + rec_sum(0)
                - 0

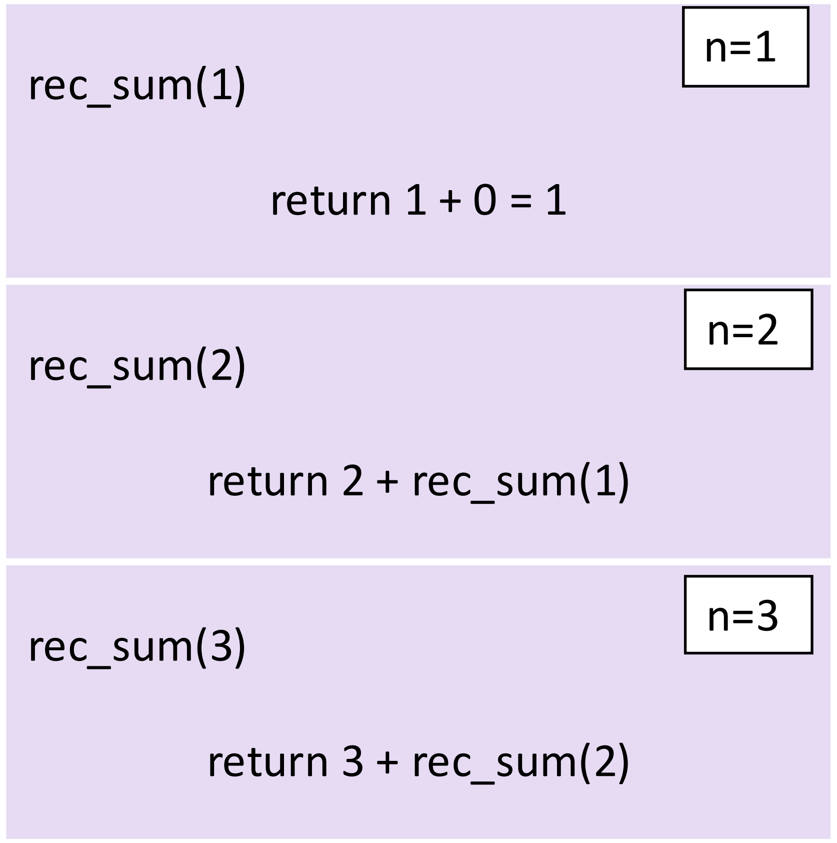


```
rec_sum(0)                              n=0
                  0

rec_sum(1)                              n=1
        return 1 + rec_sum(0)

rec_sum(2)                              n=2
        return 2 + rec_sum(1)

rec_sum(3)                              n=3
        return 3 + rec_sum(2)
```

Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

- rec_sum(3)
  - 3 + rec_sum(2)
    - 2 + rec_sum(1)
      - 1 + rec_sum(0)
        - 0
      - 1 + 0 = 1

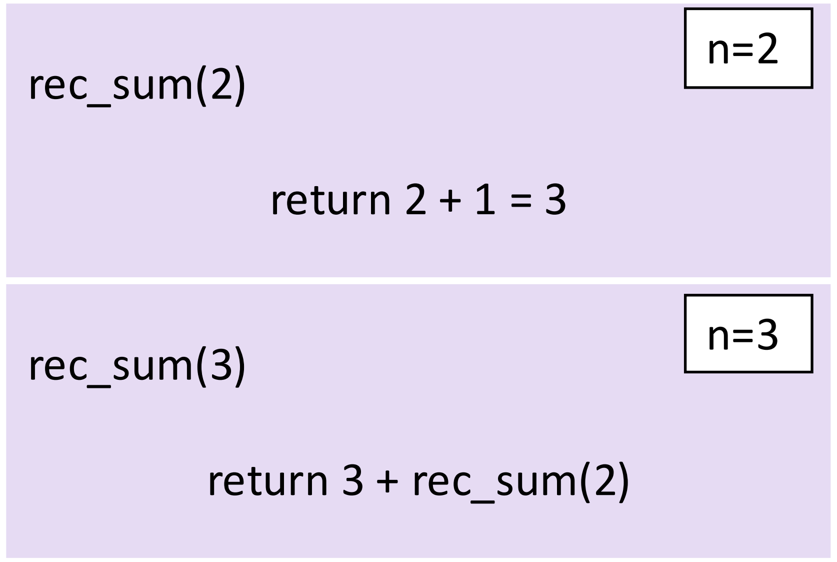


```
n=1
rec_sum(1)
    return 1 + 0 = 1

n=2
rec_sum(2)
    return 2 + rec_sum(1)

n=3
rec_sum(3)
    return 3 + rec_sum(2)
```

Answer

```
def rec_sum(n):
  if n == 0:
    return 0
  else:
    return n + rec_sum(n-1)
```

- rec_sum(3)
  - 3 + rec_sum(2)
    - 2 + rec_sum(1)
      - 1 + rec_sum(0)
        - 0
      - 1 + 0 = 1
    - 2 + 1 = 3



Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

- ➢ rec_sum(3)
    - ➢ 3 + rec_sum(2)
        - ➢ 2 + rec_sum(1)
            - ➢ 1 + rec_sum(0)
                - ➢ 0
            - ➢ 1 + 0 = 1
        - ➢ 2 + 1 = 3
    - ➢ 3 + 3 = 6

rec_sum(3) $\boxed{n=3}$

return 3 + 3 = 6

Answer

```
def rec_sum(n):
    if n == 0:
        return 0
    else:
        return n + rec_sum(n-1)
```

- rec_sum(3)
    - 3 + rec_sum(2)
        - 2 + rec_sum(1)
            - 1 + rec_sum(0)
                - 0
            - 1 + 0 = 1
        - 2 + 1 = 3
    - 3 + 3 = 6
- 6

Practice time

- Write a recursive function called `reverse` that takes a string as a parameter and reverses the string.
- Remember the steps for writing a recursive function:

1. Define what the function the name and parameters of the function are.
2. Define the recursive case
3. Define the base case
4. Put it all together

Answer

1. Define what the function the name and parameters of the function are.
   - def reverse(st)
2. Define the recursive case
   - Pretend like we have a function called reverse that we can use but only on smaller strings
   - To reverse a string:
     - remove the first character,
     - reverse the remaining characters,
     - put that first character at the end
   - The recursive relationship is:
     - reverse(st) = reverse(st[1:]) + st[0]

Answer

3. Define the base case

  - in each case, the string is getting shorter.
  - Eventually, it will be an empty string.

- 4. Put it all together

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

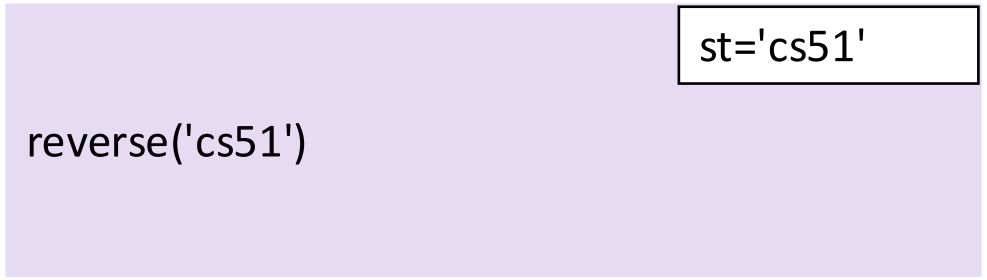
Could have also worked the other direction,

```
return st[-1] + reverse(st[:-1])
```

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- ➢ reverse('cs51')



Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
  - reverse('s51') + 'c'

st='cs51'

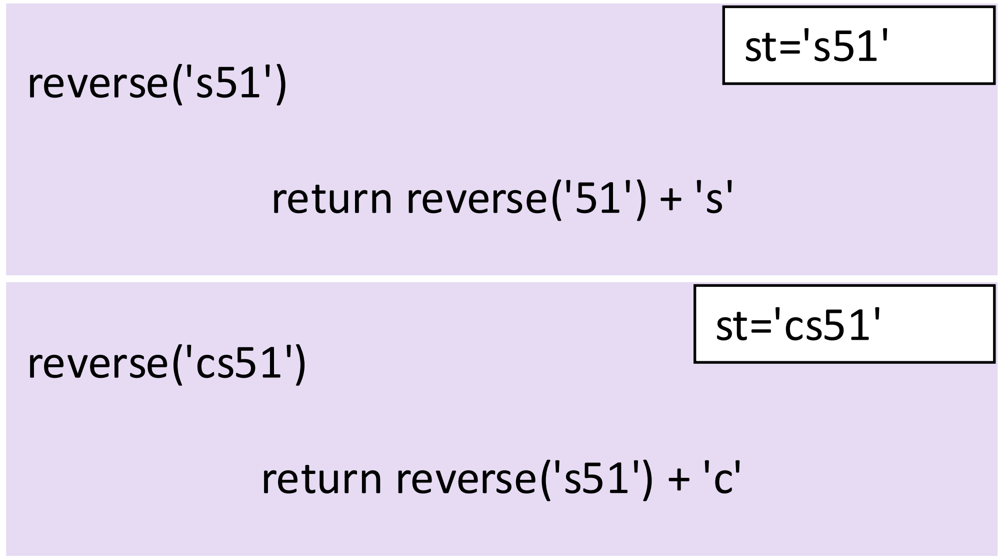
reverse('cs51')

return reverse('s51') + 'c'

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
  - reverse('s51') + 'c'
    - reverse('51') + 's'



st='s51'

reverse('s51')

return reverse('51') + 's'

st='cs51'

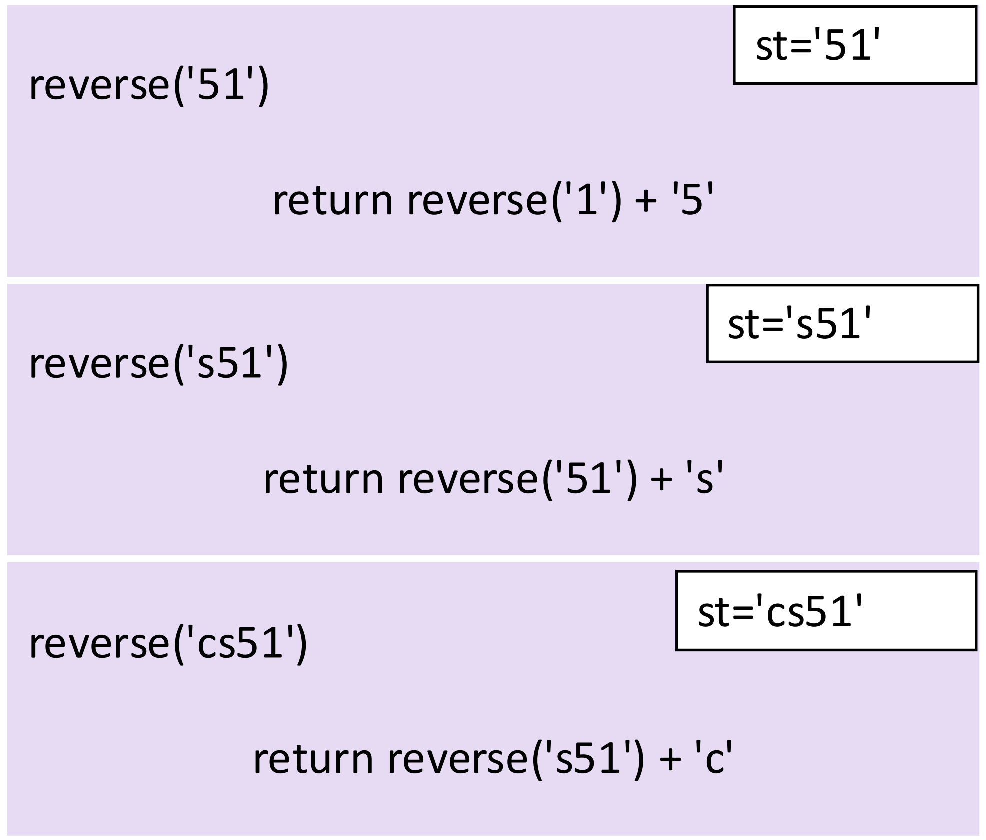
reverse('cs51')

return reverse('s51') + 'c'

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
    - reverse('s51') + 'c'
        - reverse('51') + 's'
            - reverse('1') + '5'



st='51'

reverse('51')

return reverse('1') + '5'

st='s51'

reverse('s51')

return reverse('51') + 's'

st='cs51'

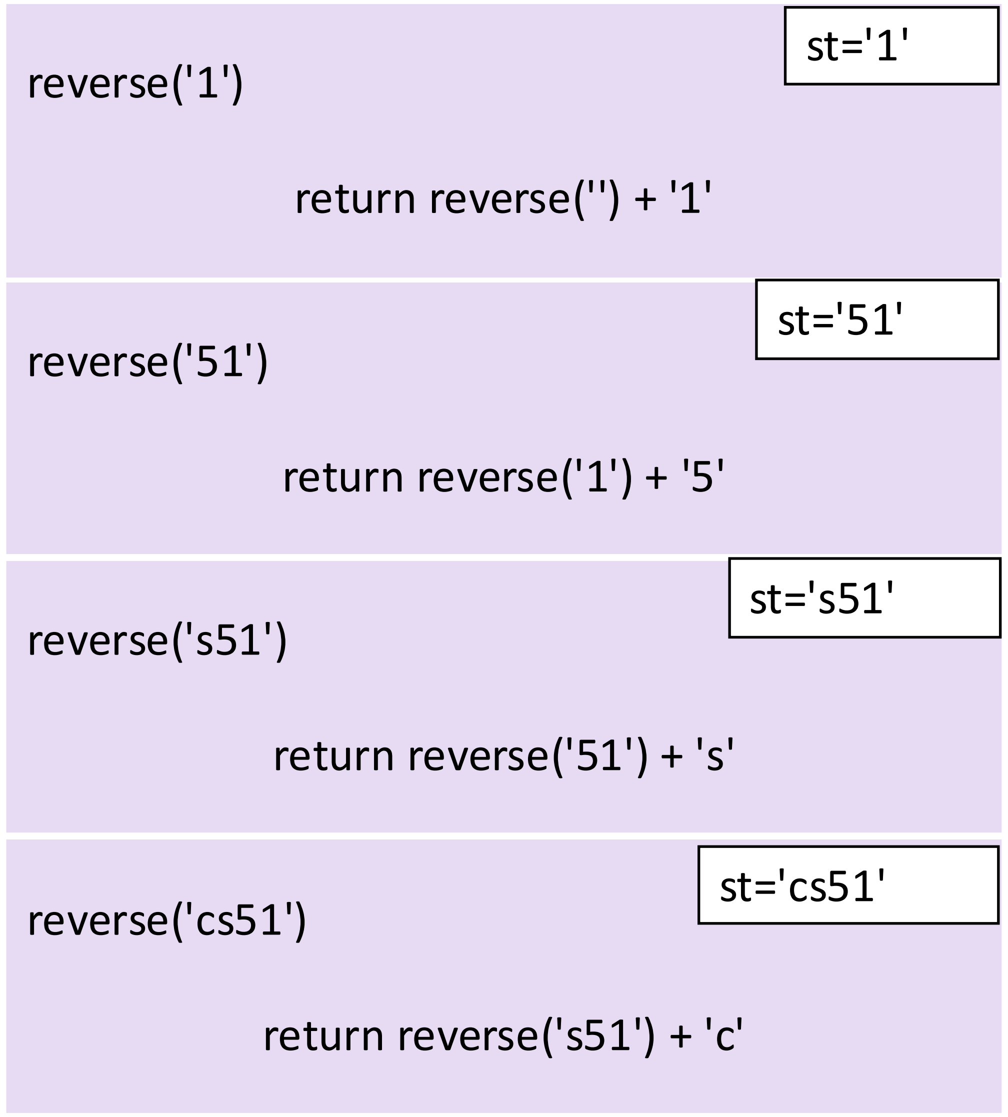
reverse('cs51')

return reverse('s51') + 'c'

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
  - reverse('s51') + 'c'
    - reverse('51') + 's'
      - reverse('1') + '5'
        - reverse('') + '1'

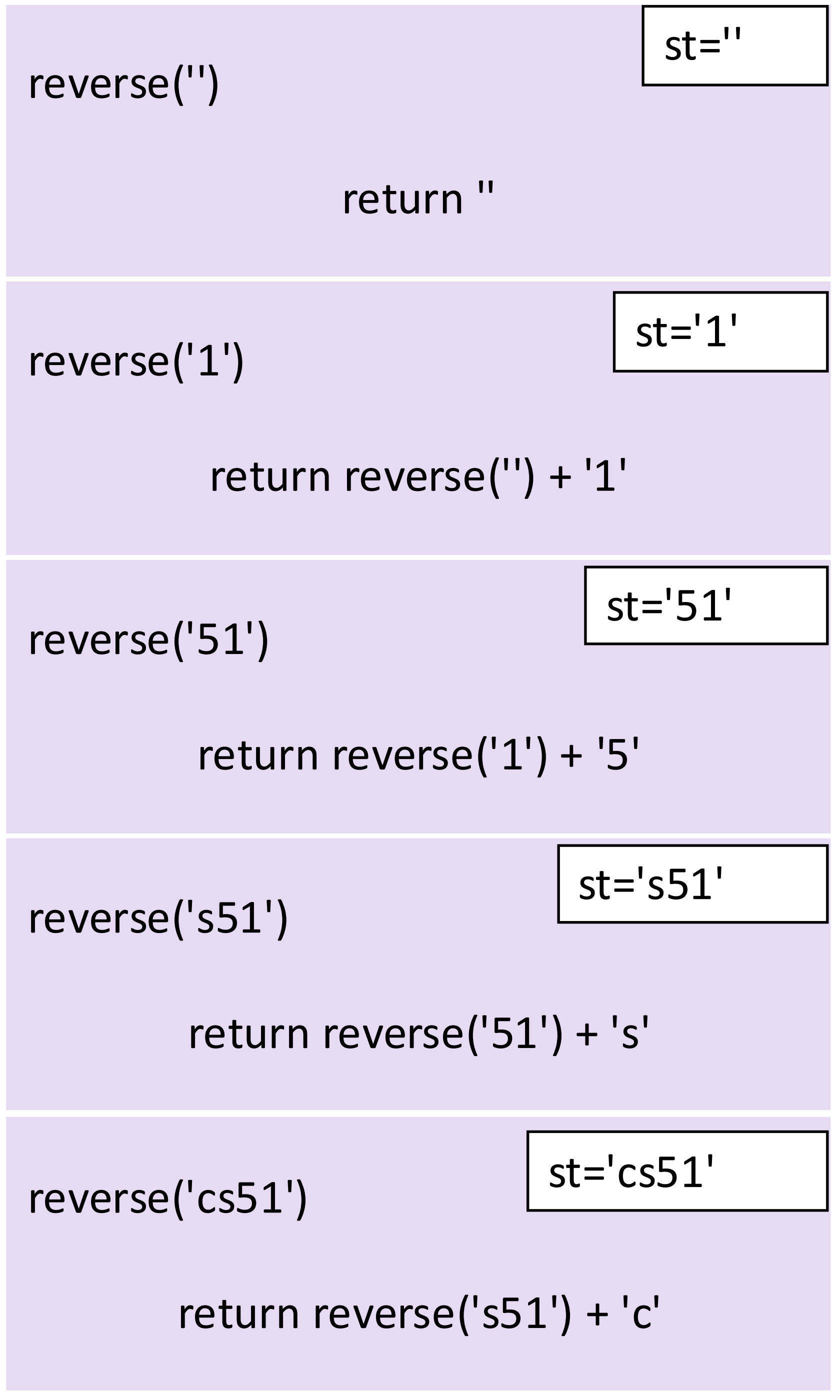


```
reverse('1')                      st='1'
        return reverse('') + '1'

reverse('51')                     st='51'
        return reverse('1') + '5'

reverse('s51')                    st='s51'
        return reverse('51') + 's'

reverse('cs51')                   st='cs51'
        return reverse('s51') + 'c'
```

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
  - reverse('s51') + 'c'
    - reverse('51') + 's'
      - reverse('1') + '5'
        - reverse('') + '1'
          - ''

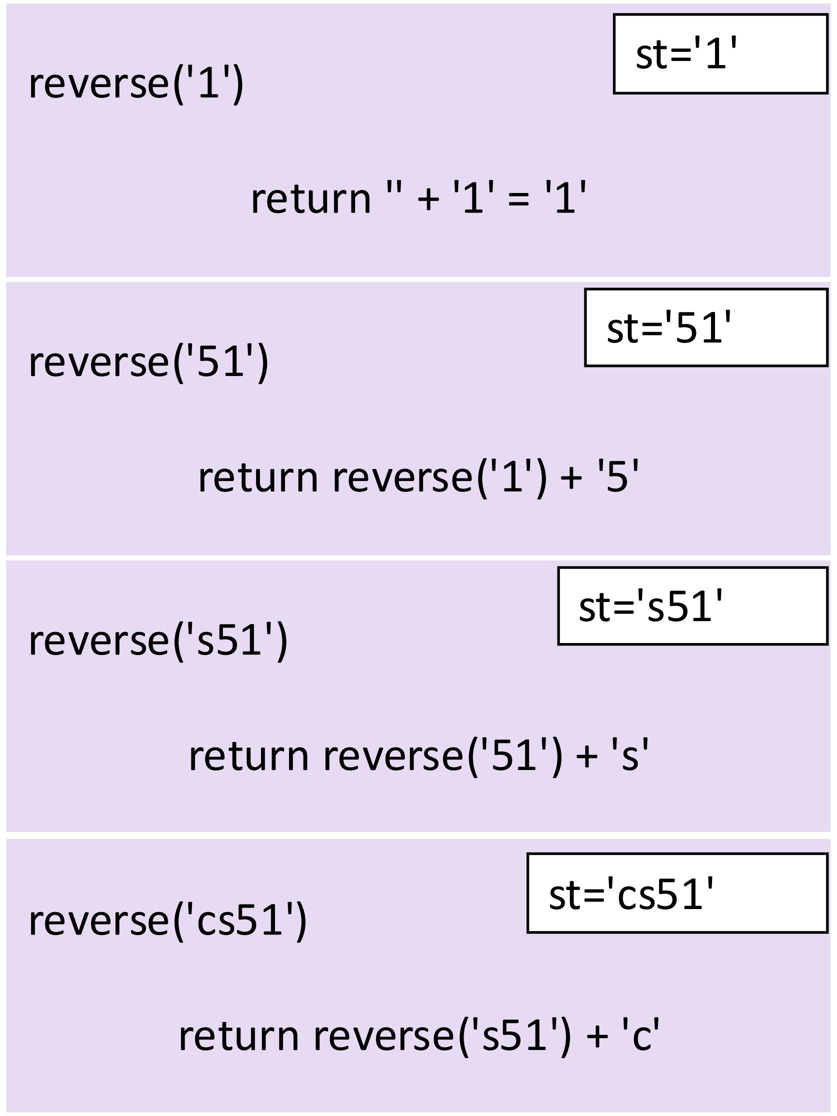


```
reverse('')                st=''
        return ''

reverse('1')               st='1'
        return reverse('') + '1'

reverse('51')              st='51'
        return reverse('1') + '5'

reverse('s51')             st='s51'
        return reverse('51') + 's'

reverse('cs51')            st='cs51'
        return reverse('s51') + 'c'
```

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

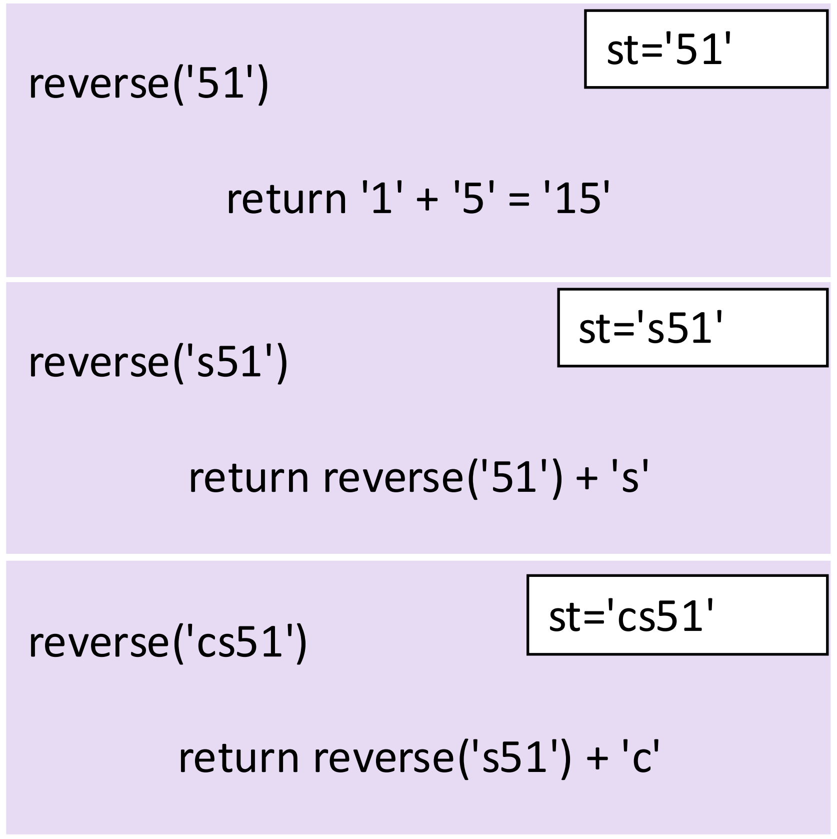
- reverse('cs51')
  - reverse('s51') + 'c'
    - reverse('51') + 's'
      - reverse('1') + '5'
        - reverse('') + '1'
          - ''
        - '' + '1' = '1'



Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
    - reverse('s51') + 'c'
        - reverse('51') + 's'
            - reverse('1') + '5'
                - reverse('') + '1'
                    - ''
                - '' + '1' = '1'
            - '1' + '5' = '15'



```
reverse('51')                st='51'
        return '1' + '5' = '15'

reverse('s51')               st='s51'
        return reverse('51') + 's'

reverse('cs51')              st='cs51'
        return reverse('s51') + 'c'
```

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
    - reverse('s51') + 'c'
        - reverse('51') + 's'
            - reverse('1') + '5'
                - reverse('') + '1'
                    - ''
                - '' + '1' = '1'
            - '1' + '5' = '15'
        - '15' + 's' = '15s'

reverse('s51') st='s51'

return '15' + 's' = '15s'

reverse('cs51') st='cs51'

return reverse('s51') + 'c'

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
  - reverse('s51') + 'c'
    - reverse('51') + 's'
      - reverse('1') + '5'
        - reverse('') + '1'
          - ''
        - '' + '1' = '1'
      - '1' + '5' = '15'
    - '15' + 's' = '15s'
  - '15s' + 'c' = '15sc'

reverse('cs51')

st='cs51'

return '15s' + 'c' = '15sc'

Answer

```
def reverse(st):
    if st == '':
        return ''
    else:
        return reverse(st[1:]) + st[0]
```

- reverse('cs51')
  - reverse('s51') + 'c'
    - reverse('51') + 's'
      - reverse('1') + '5'
        - reverse('') + '1'
          - ''
        - '' + '1' = '1'
      - '1' + '5' = '15'
    - '15' + 's' = '15s'
  - '15s' + 'c' = '15sc'
- '15sc'

st='cs51'

Practice time

- Write a recursive function `rec_binary` that takes a positive number in decimal form and returns the string representation of its binary form (unsigned).
- For example, `rec_binary(13)` should return the string '1101'
- Remember the steps:

1. Define what the function the name and parameters of the function are.
2. Define the recursive case
3. Define the base case
4. Put it all together

Answer

1. Define what the function the name and parameters of the function are.
   - def rec_binary(n)
2. Define the recursive case
   - rec_binary(n) = rec_binary(n//2) + str(n%2)

- 3. Define the base case
  - if n == 0, then the string would be '0'
  - if n == 1, then the string would be '1'

Answer

- 4. Put it all together

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)

n=5

rec_binary(5)

Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)
  - rec_binary(2) + '1'

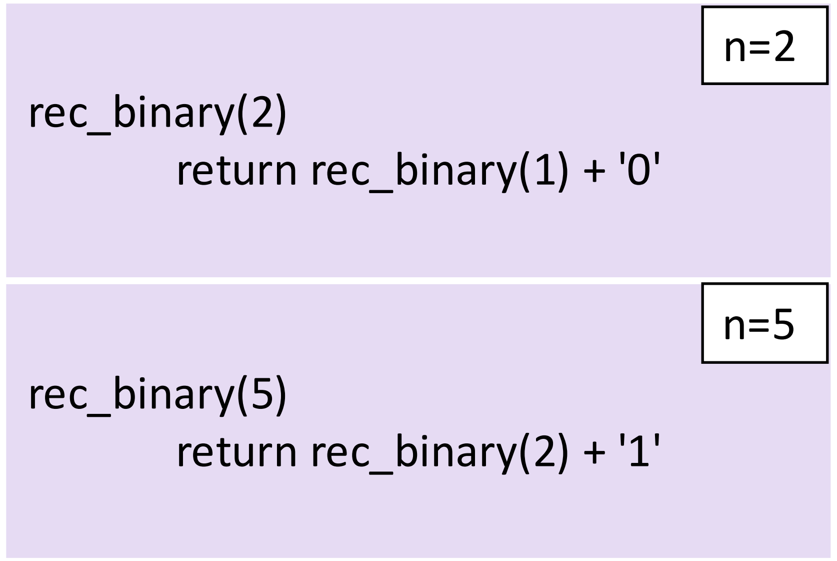
n=5

```
rec_binary(5)
        return rec_binary(2) + '1'
```

Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)
  - rec_binary(2) + '1'
    - rec_binary(1) + '0'


n=2

```
rec_binary(2)
        return rec_binary(1) + '0'
```

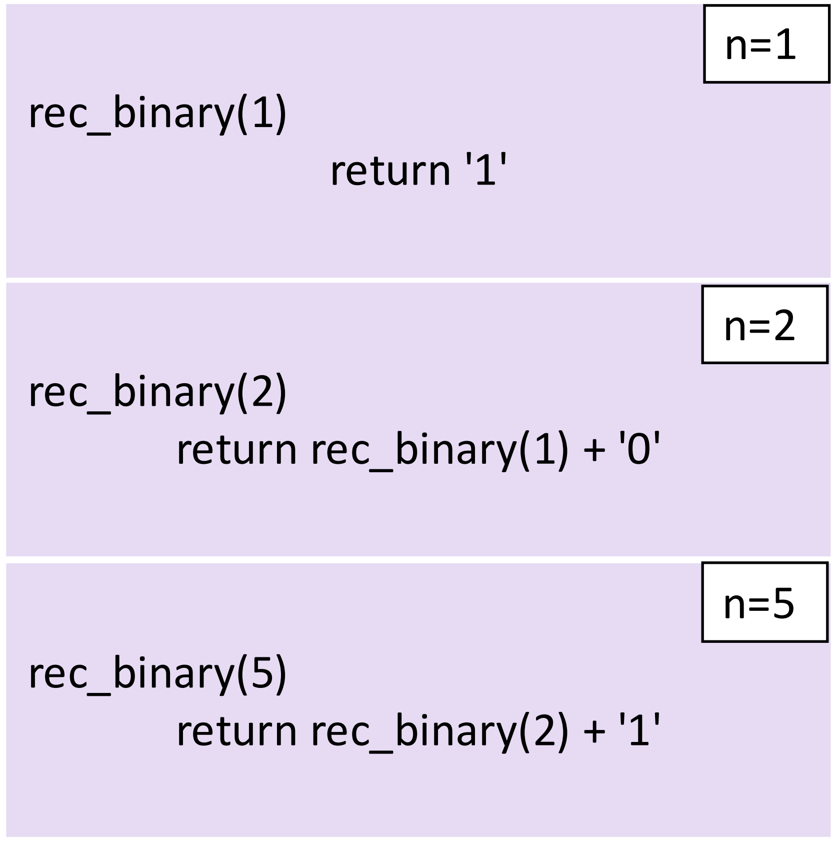
n=5

```
rec_binary(5)
        return rec_binary(2) + '1'
```

Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)
  - rec_binary(2) + '1'
    - rec_binary(1) + '0'
      - '1'



n=1

```
rec_binary(1)
                return '1'
```

n=2

```
rec_binary(2)
        return rec_binary(1) + '0'
```

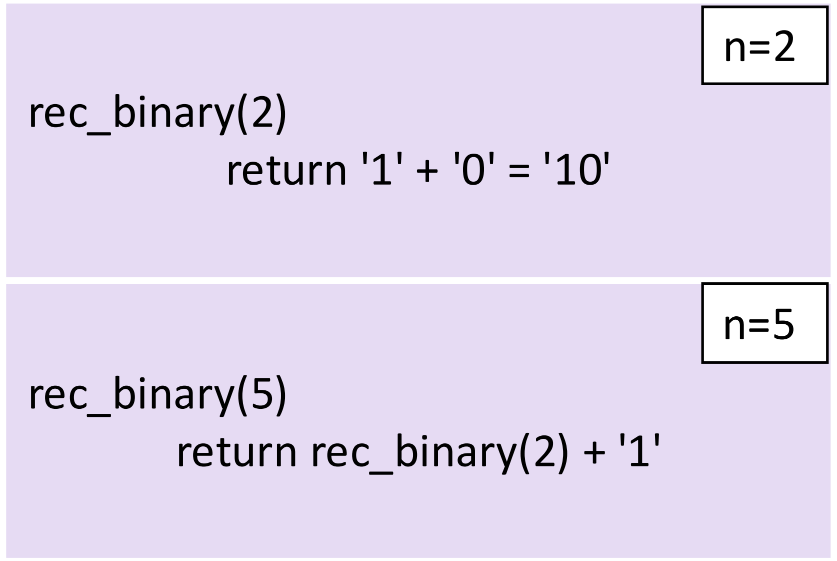
n=5

```
rec_binary(5)
        return rec_binary(2) + '1'
```

Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)
    - rec_binary(2) + '1'
        - rec_binary(1) + '0'
            - '1'
        - '1' + '0' = '10'



Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)
  - rec_binary(2) + '1'
    - rec_binary(1) + '0'
      - '1'
    - '1' + '0' = '10'
  - '10' + '1' = '101'

n=5

rec_binary(5)
    return '10' + '1' = '101'

Answer

```
def rec_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    return rec_binary(n // 2) + str(n % 2)
```

- rec_binary(5)
    - rec_binary(2) + '1'
        - rec_binary(1) + '0'
            - '1'
        - '1' + '0' = '10'
    - '10' + '1' = '101'
- '101'

Practice time

- Write a recursive function called `sum_of_digits` that takes a number and returns the sum of its digits.
- For example, `sum_of_digits(1234)` would return 10.

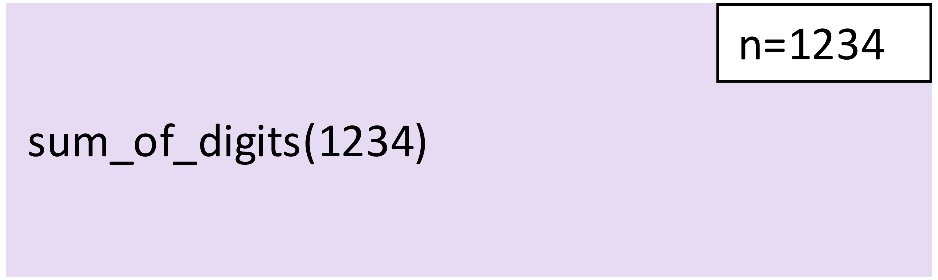
Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)



Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
    - 4 + sum_of_digits(123)

n=1234

```
sum_of_digits(1234)
        return 4 + sum_of_digits(123)
```

Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
    - 4 + sum_of_digits(123)
        - 3 + sum_of_digits(12)

n=123

sum_of_digits(123)
    return 3 + sum_of_digits(12)

n=1234

sum_of_digits(1234)
    return 4 + sum_of_digits(123)

Answer

```
def sum_of_digits(n):
  if n < 10:
    return n
  else:
    return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
  - 4 + sum_of_digits(123)
    - 3 + sum_of_digits(12)
      - 2 + sum_of_digits(1)

```
n=12
sum_of_digits(12)
        return 2 + sum_of_digits(1)

n=123
sum_of_digits(123)
        return 3 + sum_of_digits(12)

n=1234
sum_of_digits(1234)
        return 4 + sum_of_digits(123)
```

Answer

```
def sum_of_digits(n):

    if n < 10:

        return n

    else:

        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
    - 4 + sum_of_digits(123)
        - 3 + sum_of_digits(12)
            - 2 + sum_of_digits(1)
                - 1

n=1

sum_of_digits(1)
return 1

n=12

sum_of_digits(12)
return 2 + sum_of_digits(1)

n=123

sum_of_digits(123)
return 3 + sum_of_digits(12)

n=1234

sum_of_digits(1234)
return 4 + sum_of_digits(123)

Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
  - 4 + sum_of_digits(123)
    - 3 + sum_of_digits(12)
      - 2 + sum_of_digits(1)
        - 1
      - 2 + 1 = 3

n=12

```
sum_of_digits(12)
        return 2 + 1 = 3
```

n=123

```
sum_of_digits(123)
        return 3 + sum_of_digits(12)
```

n=1234

```
sum_of_digits(1234)
        return 4 + sum_of_digits(123)
```

Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
  - 4 + sum_of_digits(123)
    - 3 + sum_of_digits(12)
      - 2 + sum_of_digits(1)
        - 1
      - 2 + 1 = 3
    - 3 + 3 = 6

n=123

sum_of_digits(123)
return 3 + 3 = 6

n=1234

sum_of_digits(1234)
return 4 + sum_of_digits(123)

Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
  - 4 + sum_of_digits(123)
    - 3 + sum_of_digits(12)
      - 2 + sum_of_digits(1)
        - 1
      - 2 + 1 = 3
    - 3 + 3 = 6
  - 4 + 6 = 10

n=1234

sum_of_digits(1234)
return 4 + 6 = 10

Answer

```
def sum_of_digits(n):
    if n < 10:
        return n
    else:
        return (n % 10) + sum_of_digits(n // 10)
```

- sum_of_digits(1234)
    - 4 + sum_of_digits(123)
        - 3 + sum_of_digits(12)
            - 2 + sum_of_digits(1)
                - 1
            - 2 + 1 = 3
        - 3 + 3 = 6
    - 4 + 6 = 10
- 10

Practice time

- What does this mystery function do?

```
def rec_mystery(l):
if len(l) == 1:
  return l[0]
else:
   m = rec_mystery(l[1:])
   if m > l[0]:
      return m
   else:
      return l[0]
```

Answer

```
def rec_mystery(l):
    if len(l) == 1:
            return l[0]
    else:
        m = rec_mystery(l[1:])
        if m > l[0]:
            return m
        else:
            return l[0]
```

- Recursive function. Work through a small example:

```
>> rec_mystery([2, 4, 3, 1])
        >> m = rec_mystery([4, 3, 1])
                >> m = rec_mystery([3, 1])
                        >> m = rec_mystery([1])
                                >> 1
                        >> compares 3 and 1 and returns 3
                >> compares 4 and 3 and returns 4
        >> compares 2 and 4 and returns 4
```

- Returns the maximum element in the list!