

Mathematical Foundations

Recursion

CS51 – Fall 2025

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)

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)
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(4)
return 4 * factorial(3)

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(3)
return 3 * factorial(2)

factorial(4)
return 4 * factorial(3)

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(2)
return 2 * factorial(1)

factorial(3)
return 3 * factorial(2)

factorial(4)
return 4 * factorial(3)

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(1)
return 1

factorial(2)
return 2 * factorial(1)

factorial(3)
return 3 * factorial(2)

factorial(4)
return 4 * factorial(3)

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(2)
return 2 * 1 = 2

factorial(3)
return 3 * factorial(2)

factorial(4)
return 4 * factorial(3)

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(3)
return 3 * 2 = 6

factorial(4)
return 4 * factorial(3)

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(4)
return 4 * 6 = 24

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)
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)
```

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 for $P(k)$
  - 3. show it will work for $P(k+1)$ (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 1 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)
```

What happens in the call stack when we call `rec_sum(3)`?

rec_sum(3)

Answer

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

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(2)

return 2 + rec_sum(1)

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(0)

0

rec_sum(1)

return 1 + rec_sum(0)

rec_sum(2)

return 2 + rec_sum(1)

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(1)

return 1 + 0 = 1

rec_sum(2)

return 2 + rec_sum(1)

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(2)

return 2 + 1 = 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)

return 3 + 3 = 6

Answer

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

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

How does this work? Let's look at an example:

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

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

Practice time

- Write a recursive function called `power` that takes a base and an exponent as parameters and returns $base^{exponent}$.
- That is, it calculates base**exponent without using the ** operator. You can assume a positive exponent.
- 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 power(base, exponent)
2. Define the recursive case
   - $base^{exponent} = base^{exponent-1} * base$

3. Define the base case
   - in each case, the exponent is getting smaller.
   - Eventually, the exponent will be 0.
     - $base^{0}=1$

4. Put it all together

```
def power(base, exponent):
    if exponent == 0:
        return 1
    else:
        return base * power(base, exponent - 1)
```

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)
```

How does this work? Let's look at an example:

```
>> 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
```