Recursion

October 8, 2025

CSCI 145 – Fall 2025

Outline

- **Introducing recursion**
- Recursion examples
- Creating recursive solutions

Recursive functions

A **recursive function** is defined in terms of itself.

```
def countdown(n):
    if n > 0:
        print(n)
        countdown(n-1)
```

Outline

- Introducing recursion
- **Recursion examples**
- Creating recursive solutions

Example: countdown

```
def countdown(n):
    """
    Print numbers from n down to 1

    params: n (int) positive number
    return: None
    """
```

What is the **base case**?

if n is 0, do nothing

What is the **recursive case**?

print(n) and then call countdown(n-1)

```
>>> countdown(3)
 3
 2
 1
```

Example: Count down from n to 1

```
def countdown(n):
    """
    Print numbers from n down to 1

    params: n (int) positive number
    return: None
    """
    if n == 0:
        pass  # keyword that means "do nothing"
    else:
        print(n)
        countdown(n-1)
```

Base case

Recursive case

Example: Factorial

```
def factorial(n):
    """
    Calculate the factorial of n
    n! = n * (n-1) * ... * 1

    params: n (int)
    return: (int)
    """
```

What is the **base case**?

if n == 0, return 1

What is the **recursive case**?

return n * the factorial of n-1

```
>>> factorial(3)
6
>>> factorial(5)
120
```

Example: Factorial

```
def factorial(n):
    """
    Calculate the factorial of n
    n! = n * (n-1) * ... * 1 = n * (n-1)!

    params: n (int)
    return: (int)
    """
    if n <= 1:
        return 1
    else:
        return n * factorial(n-1)
```

Base case

Recursive case

Example: Palindrome checker

```
def is_palindrome(text):
    """
    Check if the input is a palindrome
   (the same forwards and backwards)

    params: text (str)
    return: (bool)
    """
```

What is the **base case**?

if the length of the string is less than two, it is a palindrome

What is the **recursive case**?

The string is a palindrome if the first letter and the last letter match, and the text between them is a palindrome

```
>>> is_palindrome("racecar")
True
>>> is_palindrome("frog")
False
```

Example: Palindrome checker

```
def is_palindrome(text):
    """
    Check if the input is a palindrome
    (the same forwards and backwards)

    params: text (str)
    return: (bool)
    """
    if len(text) < 2:
        return True
    else:
        return text[0] == text[-1] and is_palindrome(text[1:-1])
```

Base case

Recursive case

Outline

- Introducing recursion
- Recursion examples
- **Creating recursive solutions**

Creating a recursive solution

- **Base case**
  - A trivial version of the problem we can easily solve

- **Recursive case**
  - Decompose the problem into pieces that are either trivial to solve, or are *smaller* versions of the same problem
    - A smaller version has smaller values, less data or fewer choices
    - Reducing the problem size should make progress towards the base case
  - Compose the solutions to get a solution to the current problem

Tips for creating a recursive solution

- Do **not** try to trace down through the recursive steps
  - If you try unpacking the recursion in your head, you will start trying to write that into your code
- Do not think of the base case as the end of the recursion
  - Think of it as an easy, but valid thing to pass to the function
- Have faith that your recursive solution works on smaller problems
  - Treat it as a reliable function that does exactly what it says it does
  - Imagine someone has handed you the solution to the smaller problem – you just need to figure out how to use it

Tips for creating a recursive solution

- Start with the abstraction and write your docstring first
  - This determines the ***interface*** -- when given these inputs, this is what this function outputs
    - The interface is a form of contract you have with everyone who uses your function (included for recursive calls)
  - When you *write* the function, fulfill the contract for all cases