# More Recursion

October 13, 2025

CSCI 145 – Fall 2025

# Outline

- **Recursion refresher**
- Pending operations
- Recursion pitfalls
- Drawing with the turtle

# 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

- 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

# 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**

# Exercise: can we use recursion?

Think about some functions we have seen in this class so far. Would recursion help to approach them?

distance

Find the distance between two points

pow

Raise a number to a power

lower

Convert a string to lowercase

username_generator

Convert a name into a username

# Outline

- Recursion refresher
- **Pending operations**
- Recursion pitfalls
- Drawing with the turtle

# 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**

Pending operation that happens after the recursive call

# 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**

Pending operation that happens after the recursive call

# Pending operations after the recursive call

**Base case**

**Recursive case**

```
def go_back(n):
    if n == 0:
        print("Stop", n)
    else:
        print("Go", n)
        go_back(n-1)
        print("Back", n)
```

Pending operation that happens after the recursive call

# Outline

- Recursion refresher
- Pending operations
- **Recursion pitfalls**
- Drawing with the turtle

# Fibonacci Sequence

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

# Exercise: Recursive Fibonacci Implementation

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

```
def fib(n):
    if ____________________:
        return ____________________
    else:
        return ____________________
```

# Aside: Timing code

The `time` module has a number of ways to deal with time. `time.perf_counter()` returns the time in fractions of a second since some unknown start point. If we grab its value before and after a long computation, we can know how many seconds it took.

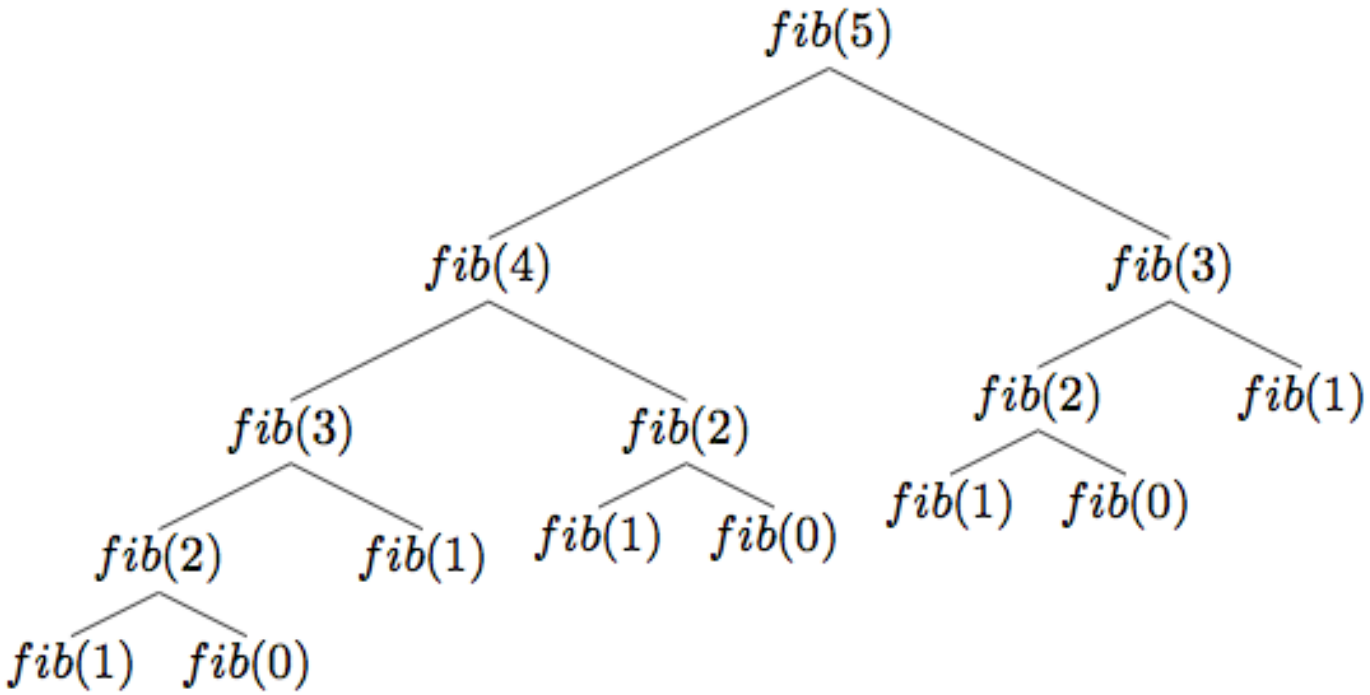
```
import time

start = time.perf_counter()

# some long process

end = time.perf_counter()
print("Execution took", end - start, "seconds")
```

# Expansion of fib(5)



# Outline

- Recursion refresher
- Pending operations
- Recursion pitfalls
- **Drawing with the turtle**

# Turtle commands

| Command | Description |
|---|---|
| forward(distance) | Move the turtle forward by the specified distance in the direction it is facing. |
| backward(distance) | Move the turtle backwards by the specified distance. |
| right(angle) | Turn the turtle to its right by the specified angle. |
| left(angle) | Turn the turtle to its left by the specified angle. |
| goto(x, y) | Move the turtle immediately to the specified location. |
| down() | Put the pen down so that the movement of the turtle leaves a track. |
| up() | Pick the pen up so that there are no tracks when the turtle moves. |
| pencolor(colorstring) | Change the color of the pen that the turtle is using. |
| tracer(state) | Turn on (True) or off (False) watching the turtle trace out the shapes. |
| update() | Show everything the turtle has drawn so far. |
| window_width() | Returns the current width of the turtle window. |
| window_height() | Returns the current height of the turtle window. |