# Recursion, Continued

Guttag Chapter 6

# Goals

1. Review concept of recursion introduced last class
2. Discuss fibonacci algorithm
3. Explore live code with recursive algorithms

# 1. Review

# Definition: Recursion

Applying the **same logic repeatedly** to solve a problem

- The problem **progresses** on each repetition

Problem-solving process stops when **base-case** is reached

# Definition: Recursion

Applying the **same logic repeatedly** to solve a problem

- The problem **progresses** on each repetition

Problem-solving process stops when **base-case** is reached

Recursively **call the function** from within the function!

**Change the input** to the function in the recursive call

Have a **conditional statement** to end the recursive calls

# Prior Recursive Examples

## Factorial Algorithm

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

## Palindrome Algorithm

```
def pal_rec(w: str) -> bool:
    if len(w) <= 1:
        return True
    elif w[0] == w[-1]:
        return pal_rec(w[1:-1])
    else:
        return False
```

# Task: Identify Key Recursive Steps

## Factorial Algorithm

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

## Palindrome Algorithm

```
def pal_rec(w: str) -> bool:
    if len(w) <= 1:
        return True
    elif w[0] == w[-1]:
        return pal_rec(w[1:-1])
    else:
        return False
```

# Solution: Identify Key Recursive Steps

## Factorial Algorithm

```
def factorial(n: int) -> int:
    if n == 1:                          BASE CASE
        return 1
    else:
        return n * factorial(n - 1)     RECURSIVE CALL
                                        PROGRESSION OF THE INPUT
```

## Palindrome Algorithm

```
def pal_rec(w: str) -> bool:
    if len(w) <= 1:                     BASE CASE
        return True
    elif w[0] == w[-1]:
        return pal_rec(w[1:-1])         RECURSIVE CALL
    else:                               PROGRESSION OF THE INPUT
        BASE CASE
        return False
```

# 2. Fibonacci Algorithm

# Fibonacci Number

Definition

- **sum of previous two** numbers in a sequence starting with 0 and 1

Example

- The zeroth fibonacci number is 0
- The first fibonacci number is 1
- The second fibonacci number is 1 (0 + 1)
- The third fibonacci number is 2 (1 + 1)
- The fourth fibonacci number is 3 (2 + 1)
- The fifth fibonacci number is 5 (3 + 2)
- etc.

In order to know the **fifth** fibonacci number, the **fourth** and the **third** must already be known!

# Fibonacci Number Algorithms

## Iterative Algorithm

```
def fib(nth: int) -> int:
    if nth <= 1:
        return nth
    zeroth = 0
    first = 1
    for _ in range(2,nth + 1):
        next = zeroth + first
        zeroth = first
        first = next
    return next
```

# Fibonacci Number Algorithms

## Iterative Algorithm

```
def fib(nth: int) -> int:
    if nth <= 1:
        return nth
    zeroth = 0
    first = 1
    for _ in range(2,nth + 1):
        next = zeroth + first
        zeroth = first
        first = next
    return next
```

## Recursive  Algorithm

```
def fib_rec(nth: int) -> int:

    if nth <= 1:

        return nth

    else:

        return fib_rec(nth-1) + fib_rec(nth-2)
```

# Fibonacci Number: Recursive Approach, details

```
def fib_rec(nth: int) -> int:
    if nth <= 1:
        return nth
    else:
        return fib_rec(nth-1) + fib_rec(nth-2)
```

base cases (recursion stops)

two recursive calls

- `fib_rec` is the call
- `n-1` is progression of the input
- `n-2` is progression of the input

# Critical Thinking

- How many recursive calls are made for the third fibonacci number?
- How many recursive calls are made for the fifth fibonacci number?
- What potential problems could arise?

# Fibonacci **Sequence**

Definition

- A sequence starting with 0 and 1, containing the **sum of previous two** numbers

Examples using list

- A seq including the zeroth fibonacci number is [0]
- A seq including the first fibonacci number is [0,1]
- A seq including the second fibonacci number is [0,1,1]
- A seq including the third fibonacci number is [0,1,1,2]
- A seq including the fourth fibonacci number is [0,1,1,2,3]
- A seq including the fifth fibonacci number is [0,1,1,2,3,5]
- etc.

In a list, the **fifth** fibonacci number can only be appended after the **fourth** and the **third** are already known!

# Fibonacci Sequence Algorithms

## Iterative Algorithm

```
def fib(nth: int) -> List[int]:
    if nth == 0:
        return [0]
    if nth == 1:
        return [0,1]
    seq = [0,1]
    for _ in range(2,nth + 1):
        next = seq[-1] + seq[-2]
        seq.append(next)
    return seq
```

# Fibonacci Sequence Algorithms

## Iterative Algorithm

```
def fib(nth: int) -> List[int]:
    if nth == 0:
        return [0]
    if nth == 1:
        return [0,1]
    seq = [0,1]
    for _ in range(2,nth + 1):
        next = seq[-1] + seq[-2]
        seq.append(next)
    return seq
```

## Recursive  Algorithm

```
def fib_rec(nth: int) -> List[int]:
    if nth == 0:
        return [0]
    if nth == 1:
        return [0,1]
    seq = fib_rec(nth - 1)
    seq.append(seq[-1] + seq[-2])
    return seq
```

# Fibonacci Sequence: Recursive Approach, details

```
def fib_rec(nth: int) -> List[int]:
    if nth == 0:
        return [0]
    if nth == 1:
        return [0,1]

    seq = fib_rec(nth - 1)

    seq.append(seq[-1] + seq[-2])
    return seq
```

base cases (recursion stops)

one recursive call

- `fib_rec` is the call
- `nth-1` is progression of the input

# Critical Thinking

Referring to the recursive algorithm,

- Why is seq initialized using the input of (nth - 1) ?
- Why would `return seq.append(seq[-1] + seq[-2])` NOT work?

```
def fib_rec(nth: int) -> List[int]:
    if nth == 0:
        return [0]
    if nth == 1:
        return [0,1]
    seq = fib_rec(nth - 1)
    seq.append(seq[-1] + seq[-2])
    return seq
```

# 3. Explore Code

# Summary

- Recursive algorithms always have
  - A recursive call
  - progression of the input
  - base case
- Fibonacci Algorithms can be recursive because each Fibonacci number depends on previous Fibonacci numbers
- Guttag Chapter 6

(Reminder that Cloning Spec Lab is due Wednesday before class)