Recursion

Guttag 6

Goals

Define Recursion

Examine Recursive Algorithms

Definition

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

- The problem **progresses** on each repetition

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

Example: Factorial

informally:

5! = 5*4*3*2*1

formally:

1! = 1

n! = n * (n - 1)!

Example: Factorial

informally:

5! = 5*4*3*2*1

formally:

1! = 1

n! = n * (n - 1)!

Iterative Algorithm

```
def factorial_iter(n: int) -> int:
    result = 1
    for i in range(1,n+1,1):
        result *= i
    return result
```

Example: Factorial

informally:

5! = 5*4*3*2*1

formally:

1! = 1

n! = n * (n - 1)!

Iterative Algorithm	Recursive Algorithm

def factorial_iter(n: int) -> int:
    result = 1
    for i in range(1,n+1,1):
        result *= i
    return result
	def factorial(n: int) -> int:
	    if n == 1:
	        return 1
	    else:
	        return n * factorial(n - 1)

Factorial: Recursive Approach, details

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

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

base case (recursion stops)

recursive call with progression of the input

- `factorial` is the call
- `n-1` is progression of the input

Critical Thinking

What happens if:

- base case is missing?
  - infinite recursive calls
- there is no progression of the input?
  - infinite recursive calls
- progression is the wrong direction?
  - infinite recursive calls

Example: Palindrome

informally:

any string with any characters that is same forward and backward

Example: Palindrome

informally:

any string with any characters that is same forward and backward

Reverse Algorithm

```
def pal_rev(w: str) -> bool:
    w_rev = w[::-1]
    if w_rev == w:
        return True
    else:
        return False
```

Example: Palindrome

informally:

any string with any characters that is same forward and backward

Reverse Algorithm

```
def pal_rev(w: str) -> bool:
    w_rev = w[::-1]
    if w_rev == w:
        return True
    else:
        return False
```

Recursive 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
```

Palindrome: Recursive Approach, details

```
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
```

base case (recursion stops)

recursive call with progression of the input

- `pal_rec` is the call
- `w[1:-1]` is progression of the input

another base case (recursion stops)

Critical Thinking

Why does it matter:

- if len(w) <= 1
  - all strings that are one character or zero characters are palindromes!
- w[0] == w[-1]
  - the first and last character must be the same in a palindrome
- progression is done using w[1:-1]
  - slicing creates a shallow copied string starting at python 1st index and ending at -1 index (not inclusive!)
  - we already checked that the first character == last characters, thus we have to remove them to continue checking.

Example: Fibonacci Number

informally:

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

Example: Fibonacci Number

informally:

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

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
```

Example: Fibonacci Number

informally:

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

Iterative Algorithm	Recursive Algorithm

<pre>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</pre>
	<pre>def fib(nth: int) -> int:
	
	    if nth <= 1:
	
	        return nth
	
	    else:
	
	        return fib(nth-1) + fib(nth-2)</pre>

Fibonacci: Recursive Approach, details

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

base cases (recursion stops)

two recursive calls with progression of the input

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

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

BASE CASE

RECURSIVE CALL

PROGRESSION OF THE INPUT

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
```

BASE CASE

RECURSIVE CALL

BASE CASE

PROGRESSION OF THE INPUT

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