

# CS 1110:

## Introduction to Computing Using Python

Lecture 16

### **More Recursion**

[Andersen, Gries, Lee, Marschner, Van Loan, White]

# Announcements

---

- We can't check off labs in professor office hours
- Reading for next week: Chapters 15 and 16

# Announcements: A3

---

- **Due:** Thursday, March 30<sup>th</sup>, 11:59pm
- `trigram_generation`: “REQUIREMNET [sic]: first, randomly pick a starting bigram “w1 w2”.”
- This means, “pick “w1 w2” randomly from the sample text, just like you picked a unigram from the text in `bigram_generation`.”

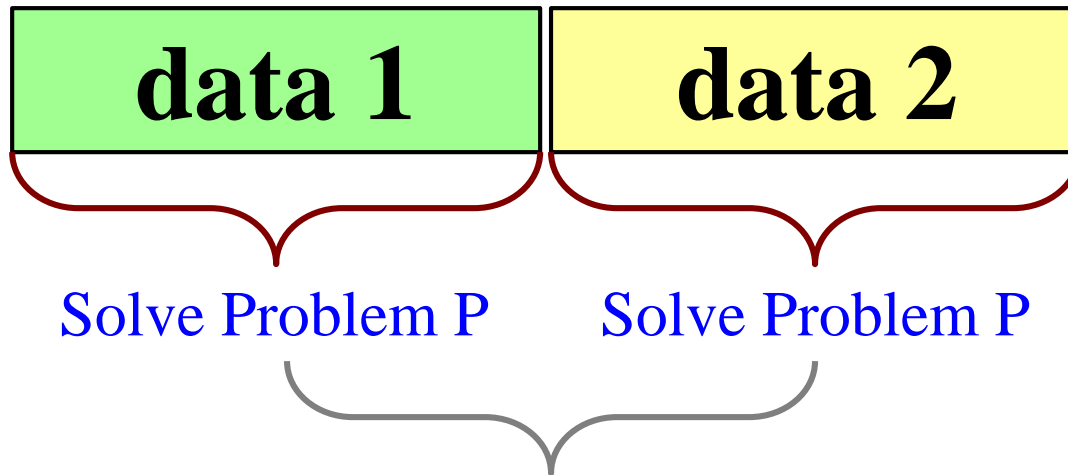
# Recall: Divide and Conquer

---

**Goal:** Solve problem P on a piece of data



**Idea:** Split data into two parts and solve problem



**Combine Answer!**

# Example: Reversing a String

---

```
def reverse(s):
```

```
    """Returns: reverse of s
```

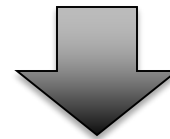
```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

```
    # 2. Break into two parts
```

```
    # 3. Combine the result
```

H	e	l	l	o	!
---	---	---	---	---	---



!	o	l	l	e	H
---	---	---	---	---	---

# Example: Reversing a String

```
def reverse(s):
```

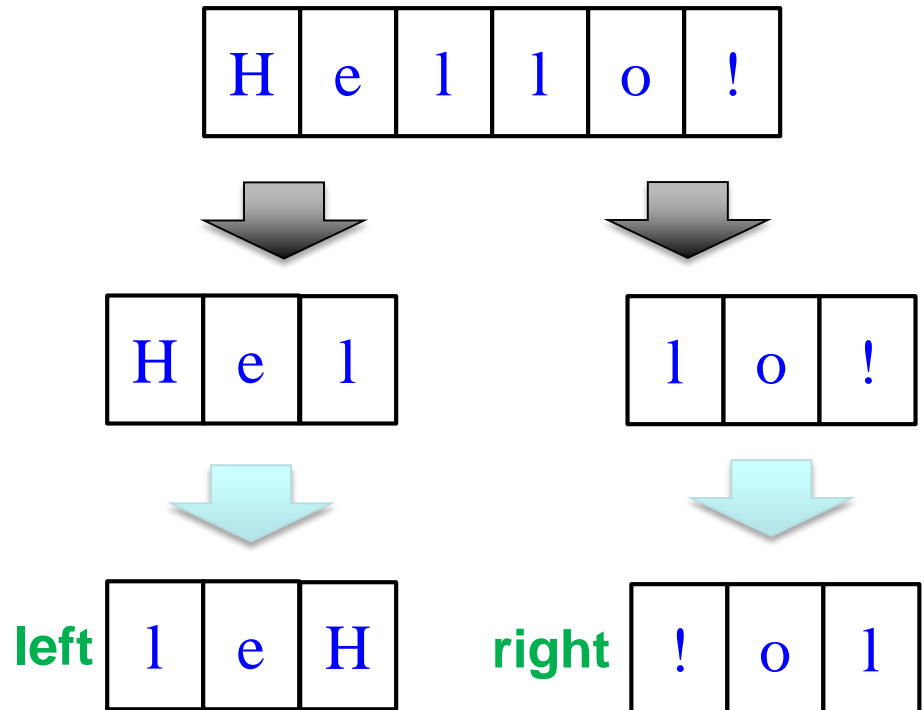
```
    """Returns: reverse of s
```

```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

```
    # 2. Break into two parts
```

```
    # 3. Combine the result
```



# Example: Reversing a String

```
def reverse(s):
```

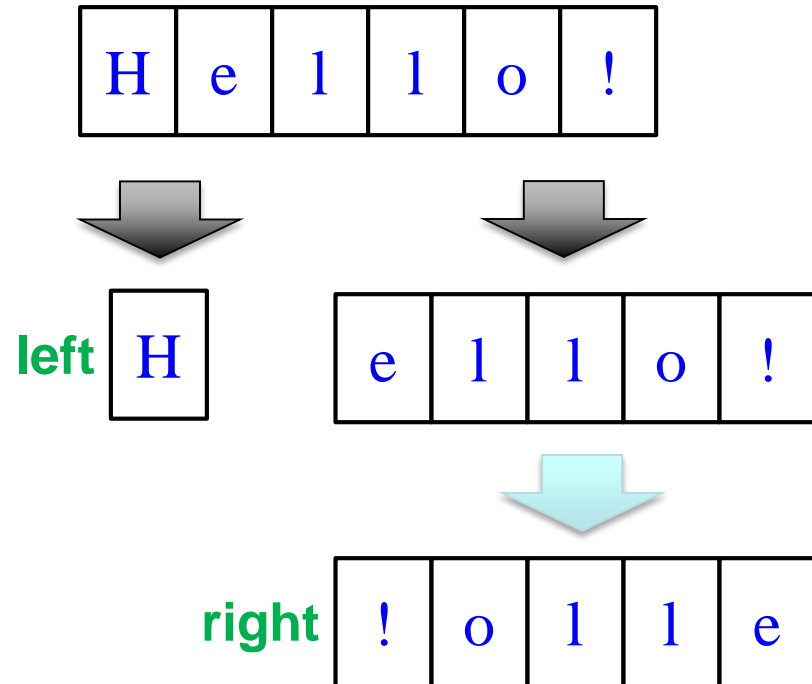
```
    """Returns: reverse of s
```

```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

```
    # 2. Break into two parts
```

```
    # 3. Combine the result
```



# Example: Reversing a String

```
def reverse(s):
```

```
    """Returns: reverse of s
```

```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

```
    # 2. Break into two parts
```

```
    # 3. Combine the result
```

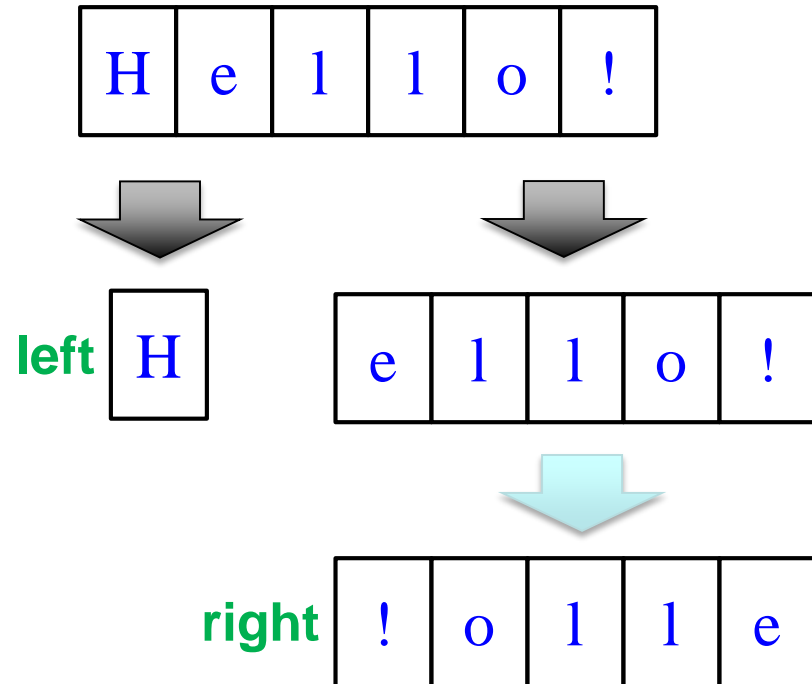
```
    return
```

```
        A: left + right
```

```
        B: right + left
```

```
        C: left
```

```
        D: right
```



**CORRECT**



# Example: Reversing a String

```
def reverse(s):
```

```
    """Returns: reverse of s
```

```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

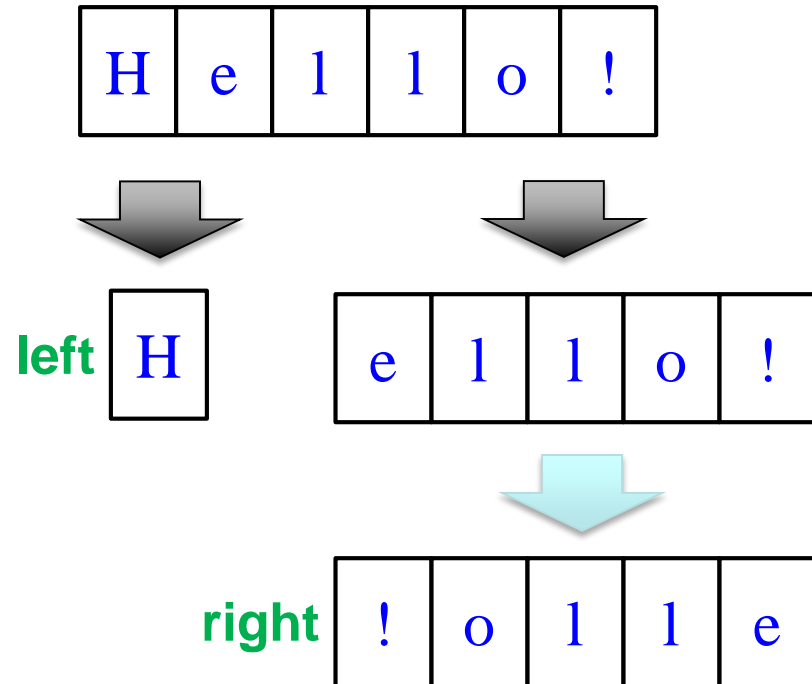
```
    # 2. Break into two parts
```

```
    left = reverse(s[0])
```

```
    right = reverse(s[1:])
```

```
    # 3. Combine the result
```

```
    return right+left
```



**Note:** This question was problematic as presented in lecture, so it has been changed a bit.

def reverse(s):

"""Returns: reverse of s

Precondition: s a string"""

# 1. Handle small data

A: if s == "":  
    return s

B: if len(s) <= 2:  
    return s

C: if len(s) <= 1:  
    return s

# 2. Break into two parts

left = reverse(s[0])

right = reverse(s[1:])

# 3. Combine the result

return right+left

H	e	l	l	o	!
---	---	---	---	---	---

**CORRECT**

D: Either A or C  
would work

E: A, B, and C  
would all work

**Note:** This question was problematic as presented in lecture, so it has been changed a bit.

```
def reverse(s):
```

```
    """Returns: reverse of s
```

```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

```
A: if s == "":  
    return s
```

```
B: if len(s) <= 2:  
    return s
```

```
C: if len(s) <= 1:  
    return s
```

```
    # 2. Break into two parts
```

```
    left = s[0]
```

```
    right = reverse(s[1:])
```

```
    # 3. Combine the result
```

```
    return right+left
```

H	e	l	l	o	!
---	---	---	---	---	---

**CORRECT**

```
D: Either A or C  
    would work
```

```
E: A, B, and C  
    would all work
```

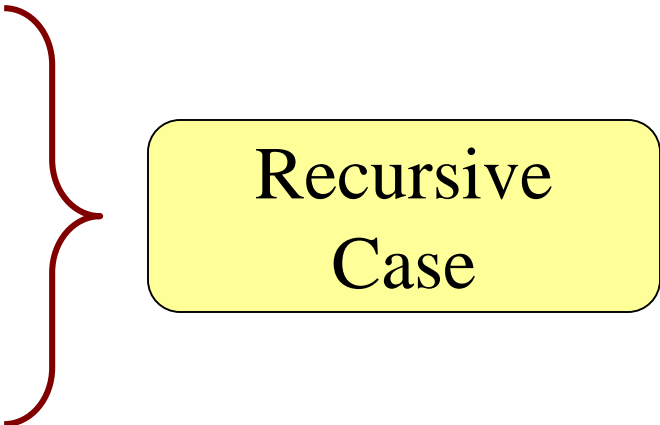
# Example: Reversing a String

---

```
def reverse(s):  
    """Returns: reverse of s  
    Precondition: s a string"""  
    # 1. Handle small data  
    if len(s) <= 1:  
        return s  
  
    # 2. Break into two parts  
    left = s[0]  
    right = reverse(s[1:])  
  
    # 3. Combine the result  
    return right+left
```



Base Case



Recursive  
Case

# Alternate Implementation

---

```
def reverse(s):  
    """Returns: reverse of s  
    Precondition: s a string"""  
    # 1. Handle small data  
    if len(s) <= 1:  
        return s  
  
    # 2. Break into two parts  
    left = reverse(s[:len(s)-1])  
    right = reverse(s[len(s)-1])  
  
    # 3. Combine the result  
    return right+left
```

Does this work?

**CORRECT**

**A: YES**

**B: NO**

# Alternate Implementation

---

```
def reverse(s):  
    """Returns: reverse of s  
    Precondition: s a string"""  
    # 1. Handle small data  
    if len(s) <= 1:  
        return s  
  
    # 2. Break into two parts  
    left = reverse(s[:2])  
    right = reverse(s[2:])  
  
    # 3. Combine the result  
    return right+left
```

Does this work?

A: YES

CORRECT B: NO

# Alternate Implementation

```
def reverse(s):
```

```
    """Returns: reverse of s
```

```
    Precondition: s a string"""
```

```
    # 1. Handle small data
```

```
    if len(s) <= 1:
```

```
        return s
```

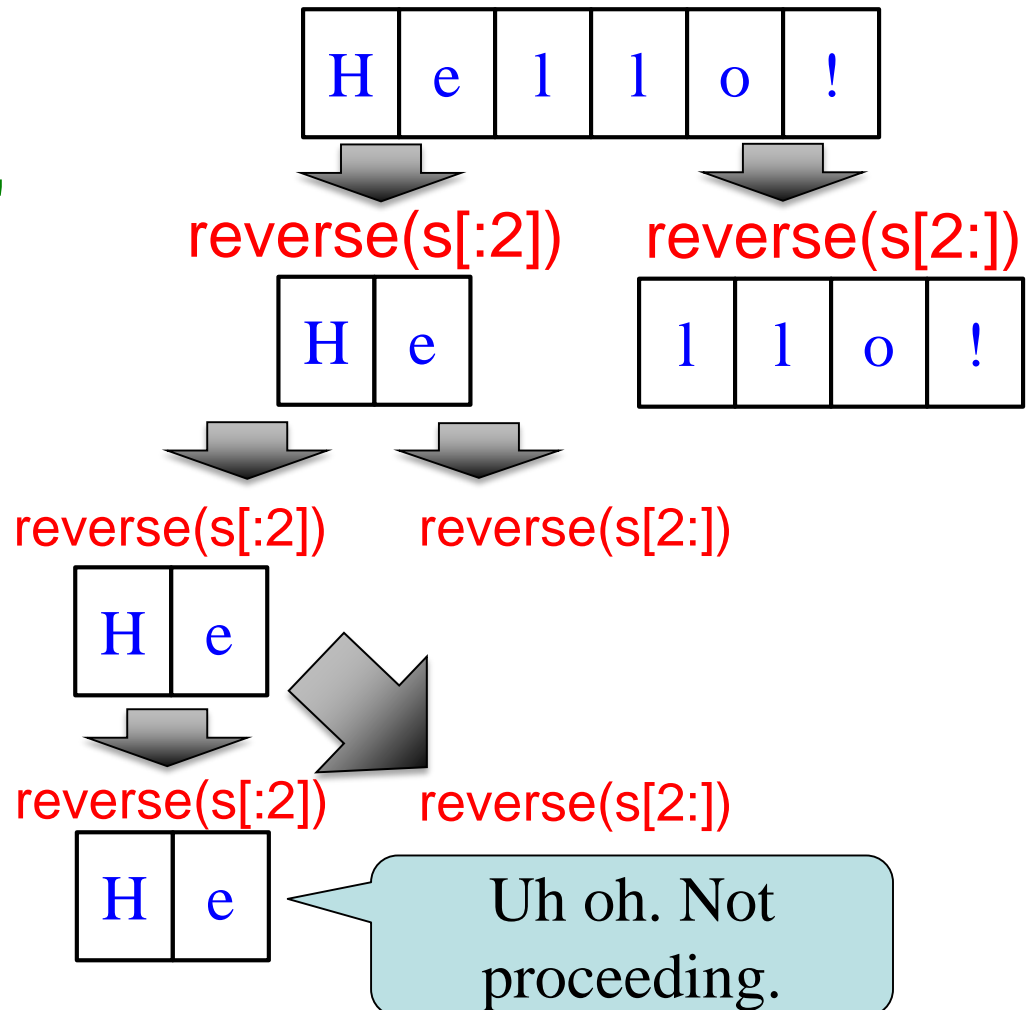
```
    # 2. Break into two parts
```

```
    left = reverse(s[:2])
```

```
    right = reverse(s[2:])
```

```
    # 3. Combine the result
```

```
    return right+left
```



# Alternate Implementation

---

```
def reverse(s):  
    """Returns: reverse of s  
    Precondition: s a string"""  
    # 1. Handle small data  
    if len(s) <= 1:  
        return s  
    if len(s) == 2:  
        return s[1] + s[0]  
  
    # 2. Break into two parts  
    left = reverse(s[:2])  
    right = reverse(s[2:])  
    # 3. Combine the result  
    return right+left
```

Does this work?

**CORRECT**

**A: YES**

**B: NO**



# Alternate Implementation

---

```
def reverse(s):  
    """Returns: reverse of s  
    Precondition: s a string"""  
    # 1. Handle small data  
    if len(s) <= 1:  
        return s  
  
    # 2. Break into two parts  
    half = len(s)/2  
    left = reverse(s[:half])  
    right = reverse(s[half:])  
  
    # 3. Combine the result  
    return right+left
```

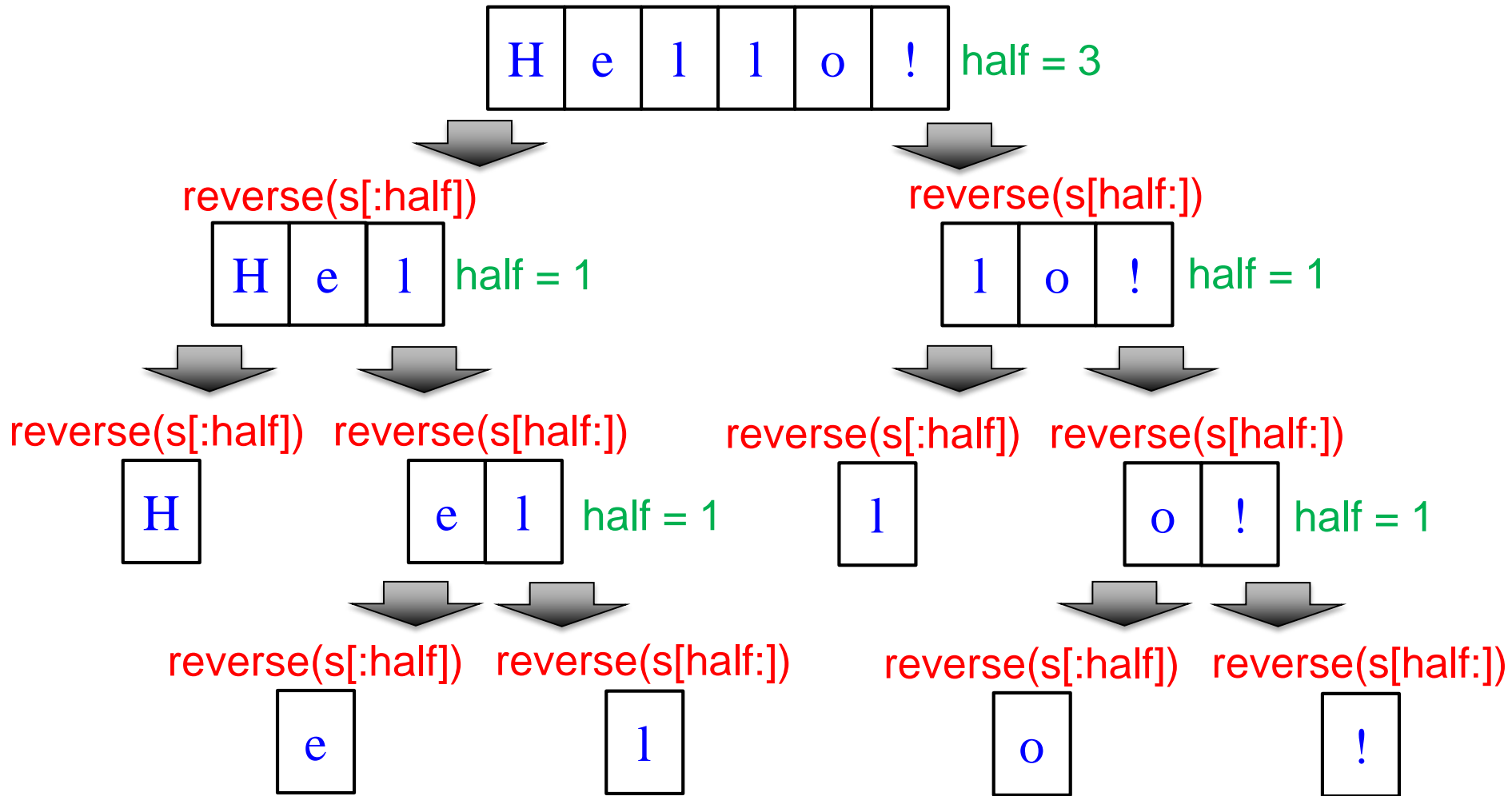
Does this work?

**CORRECT**

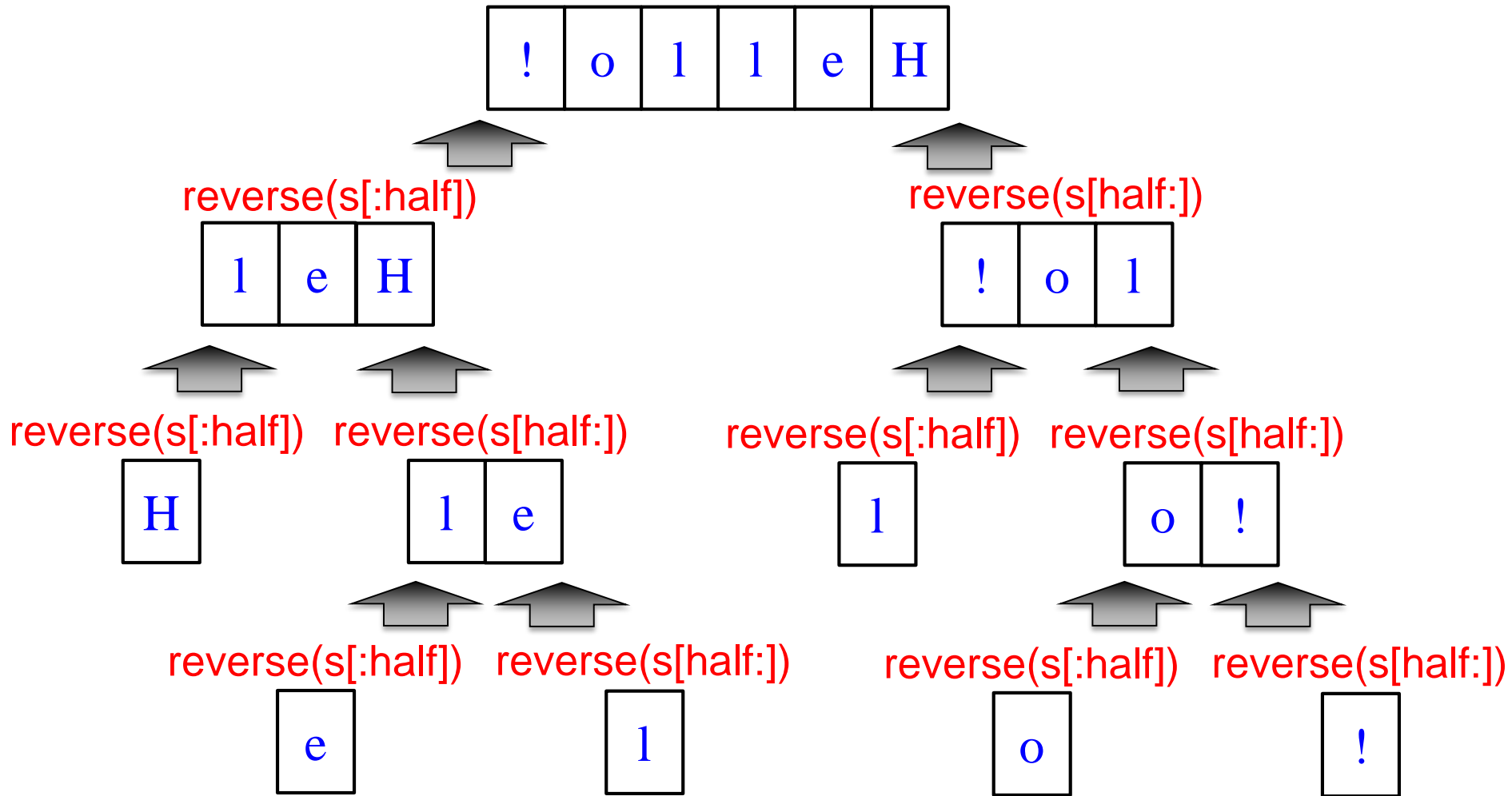
**A: YES**

**B: NO**

# Alternate Implementation



# Alternate Implementation



# Example: Palindromes

---

- **Example:**

AMANAPLANACANALPANAMA

- Can we define recursively?

# Example: Palindromes

---

- String with  $\geq 2$  characters is a palindrome if:
  - its first and last characters are equal, and
  - the rest of the characters form a palindrome

- **Example:**

have to be the same

AMANAPLANACANALPANAMA

has to be a palindrome

- **Implement:** `def ispalindrome(s):`

"""Returns: True if s is a palindrome"""

# Example: Palindromes

- String with  $\geq 2$  characters is a palindrome if:
  - its first and last characters are equal, and
  - the rest of the characters form a palindrome

```
def ispalindrome(s):
```

```
    """Returns: True if s is a palindrome"""
```

```
    if len(s) < 2:
```

```
        return True
```

**Base case**

```
    ends = s[0] == s[-1]
```

```
    middle = ispalindrome(s[1:-1])
```

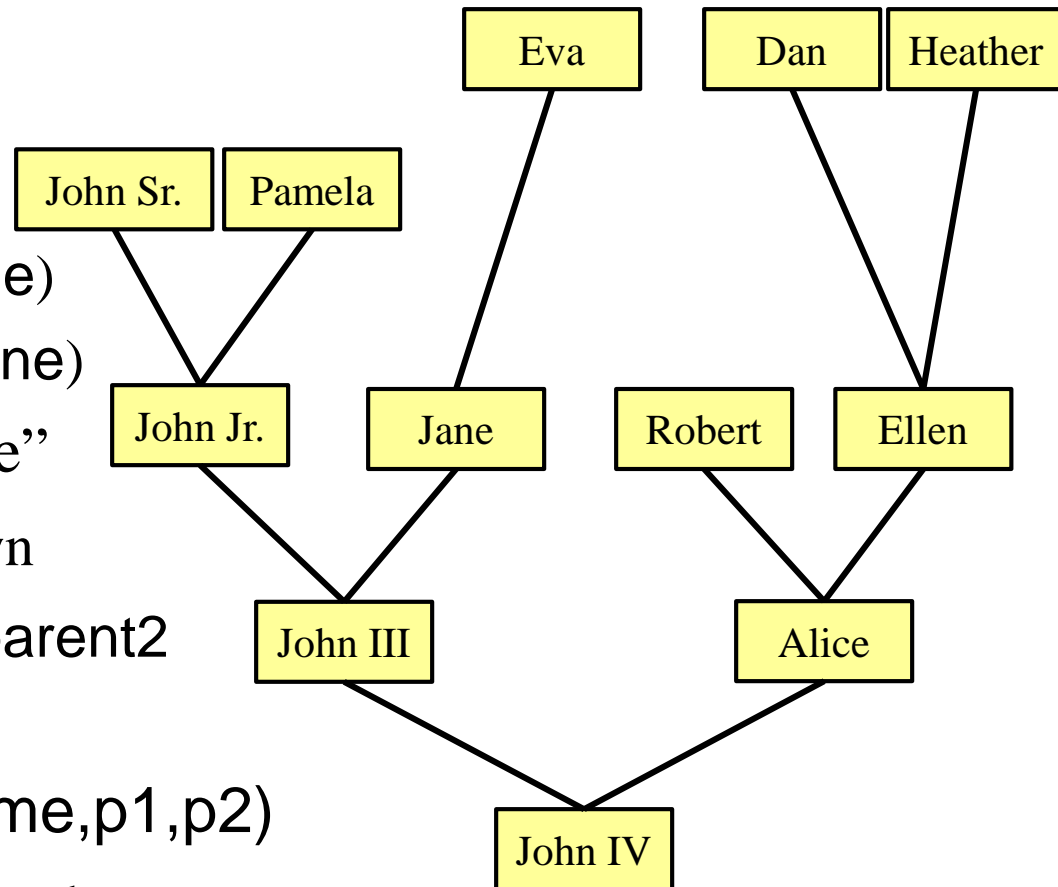
```
    return ends and middle
```

**Recursive case**

Recursive  
Definition

# Recursion and Objects

- Class Person (person.py)
  - Objects have 3 attributes
  - **name**: String
  - **parent1**: Person (or None)
  - **parent2**: Person (or None)
- Represents the “family tree”
  - Goes as far back as known
  - Attributes parent1 and parent2 are None if not known
- **Constructor**: Person(name,p1,p2)
  - Or Person(n) if no parents known

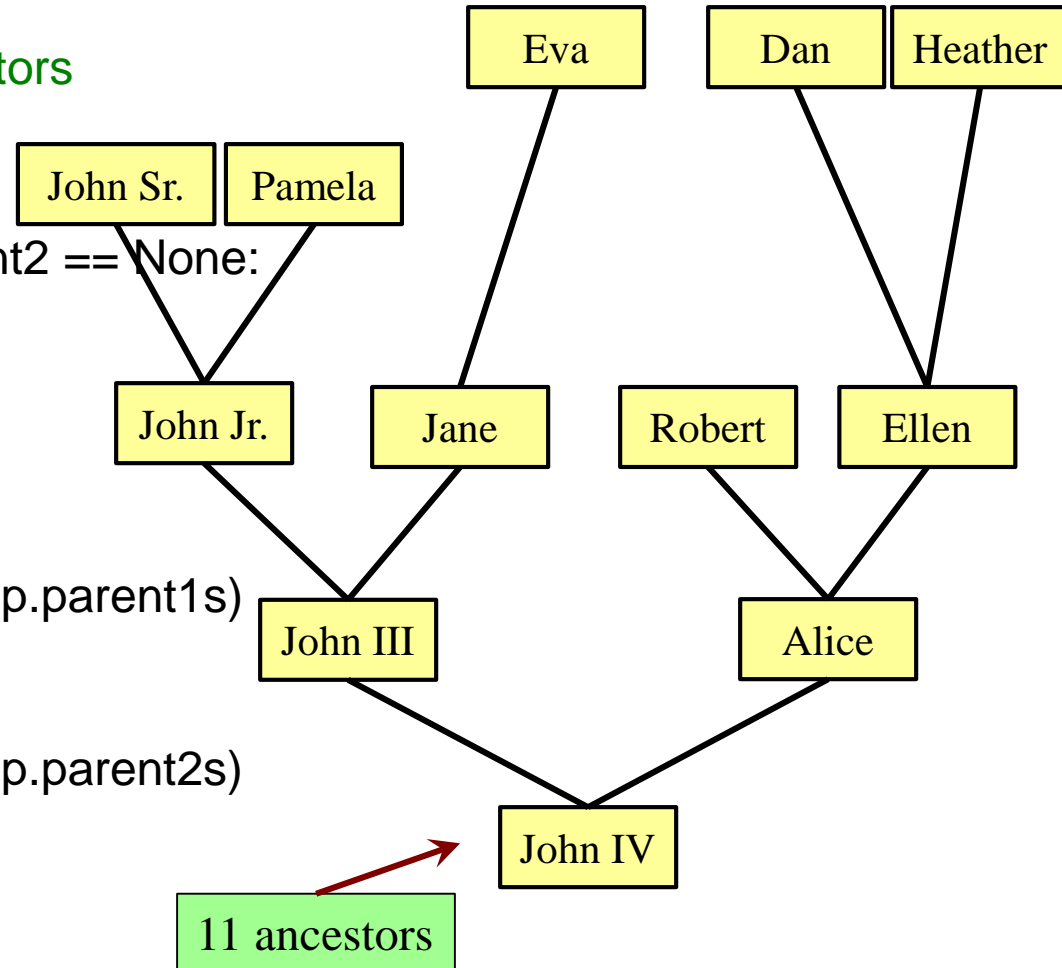







# Recursion and Objects

```
def num_ancestors(p):  
    """Returns: num of known ancestors  
    Pre: p is a Person"""  
    # 1. Handle small data.  
    if p.parent1 == None and p.parent2 == None:  
        | return 0  
  
    # 2. Break into two parts  
    parent1s = 0  
    if p.parent1 != None:  
        | parent1s = 1+num_ancestors(p.parent1s)  
    parent2s = 0  
    if p.parent2 != None:  
        | parent2s = 1+num_ancestors(p.parent2s)  
  
    # 3. Combine the result  
    return parent1s+parent2s
```



# Recursion and Objects

```
def num_ancestors(p):  
    """Returns: num of known ancestors  
    Pre: p is a Person"""  
    # 1. Handle small data.  
    if p.parent1 == None and p.parent2 == None:  
        | return 0  
  
    # 2. Break into two parts  
    parent1s = 0  
    if p.parent1 != None:  
        | parent1s = 1+num_ancestors(p.parent1s)  
    parent2s = 0  
    if p.parent2 != None:  
        | parent2s = 1+num_ancestors(p.parent2s)  
  
    # 3. Combine the result  
    return parent1s+parent2s
```

 We don't actually need this.  
It is handled by the conditionals in #2.

# Challenge: All Ancestors

```
def all_ancestors(p):
```

```
    """Returns: list of all ancestors of p"""
```

```
    # 1. Handle small data.
```

```
    # 2. Break into parts.
```

```
    # 3. Combine answer.
```

