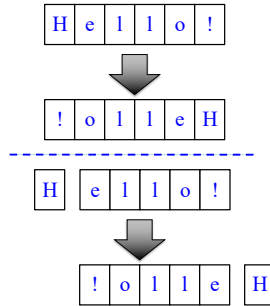


Example: Reversing a String

```
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
```

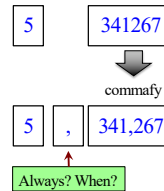


1

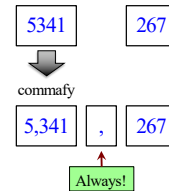
How to Break Up a Recursive Function?

```
def commafy(s):
    """Returns: string with commas every 3 digits
    e.g. commafy('5341287') = '5,341,287'
    Precondition: s represents a non-negative int"""
```

Approach 1



Approach 2



2

How to Break Up a Recursive Function?

```
def commafy(s):
    """Returns: string with commas every 3 digits
    e.g. commafy('5341287') = '5,341,287'
    Precondition: s represents a non-negative int"""
    # 1. Handle small data.
    if len(s) <= 3:
        return s
    # 2. Break into two parts
    left = commafy(s[:-3])
    right = s[-3:] # Small part on RIGHT
    # 3. Combine the result
    return left + ',' + right
```

Base Case

Recursive Case

3

How to Break Up a Recursive Function?

```
def exp(b, c)
    """Returns: b^c
    Precondition: b a float, c >= 0 an int"""
```

Approach 1

$$12^{256} = 12 \times (12^{255})$$

Recursive

$$b^c = b \times (b^{c-1})$$

Approach 2

$$12^{256} = (12^{128}) \times (12^{128})$$

Recursive Recursive

$$b^c = (b \times b)^{c/2} \text{ if } c \text{ even}$$

4

Raising a Number to an Exponent

```
def exp(b, c)
    """Returns: b^c
    Precond: b a float, c >= 0 an int"""
    # b^0 is 1
    if c == 0:
        return 1
    # c > 0
    if c % 2 == 0:
        return exp(b*b, c//2)
    return b*exp(b*b, (c-1)//2)
```

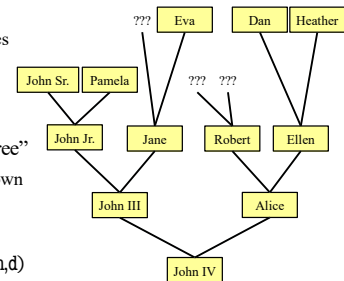
c	# of calls
0	0
1	1
2	2
4	3
8	4
16	5
32	6
2^n	n + 1

32768 is 2¹⁵
b³²⁷⁶⁸ needs only 215 calls!

5

Recursion and Objects

- Class Person (person.py)
 - Objects have 3 attributes
 - name: String
 - mom: Person (or None)
 - dad: Person (or None)
- Represents the “family tree”
 - Goes as far back as known
 - Attributes mom and dad are None if not known
- Constructor: Person(n,m,d)
 - Or Person(n) if no mom, dad



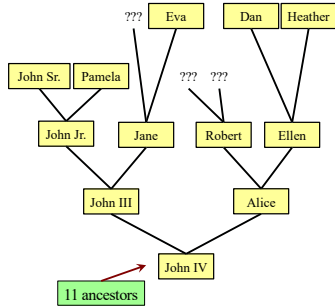
6

Recursion and Objects

```
def num_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""
    # 1. Handle small data.
    if p.mom == None and p.dad == None:
        return 0

    # 2. Break into two parts
    moms = 0
    if not p.mom == None:
        moms = 1 + num_ancestors(p.mom)
    dads = 0
    if not p.dad == None:
        dads = 1 + num_ancestors(p.dad)

    # 3. Combine the result
    return moms + dads
```



7

Example: Palindromes

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

Example: AMANAPLANACANALPANAMA

have to be the same (pointing to 'A' and 'A')
has to be a palindrome (pointing to the rest of the string)

- Function to Implement:**

```
def ispalindrome(s):
    """Returns: True if s is a palindrome"""
```

8

Example: Palindromes

- String with ≥ 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

    # Halves not the same; not divide and conquer
    ends = s[0] == s[-1]
    middle = ispalindrome(s[1:-1])
    return ends and middle
```

Recursive Definition

Base case

Recursive case

9

Recursive Functions and Helpers

```
def ispalindrome2(s):
    """Returns: True if s is a palindrome
    Case of characters is ignored"""
    if len(s) < 2:
        return True
    # Halves not the same; not divide and conquer
    ends = equals_ignore_case(s[0], s[-1])
    middle = ispalindrome2(s[1:-1])
    return ends and middle
```

Use helper functions!

- Pull out anything not part of the recursion
- Keeps your code simple and easy to follow

```
def equals_ignore_case(a, b):
    """Returns: True if a and b are same ignoring case"""
    return a.upper() == b.upper()
```

10

Example: More Palindromes

```
def ispalindrome3(s):
    """Returns: True if s is a palindrome
    Case of characters and non-letters ignored."""
    return ispalindrome2(depunct(s))
```

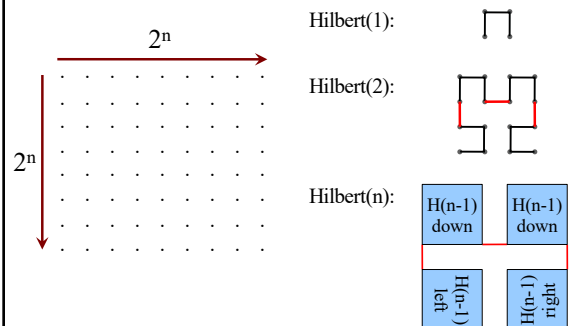
```
def depunct(s):
    """Returns: s with non-letters removed"""
    if s == "":
        return s
    # Combine left and right
    if s[0] in string.letters:
        return s[0] + depunct(s[1:])
    # Ignore left if it is not a letter
    return depunct(s[1:])
```

Use helper functions!

- Sometimes the helper is a recursive function
- Allows you break up problem in smaller parts

11

Hilbert's Space Filling Curve



12