Graphics and Computer Vision

Fruitful Recursion

Recursion reminders

Three Laws of Recursion:

- A recursive algorithm must have a **base case**.
- A recursive algorithm must **change its state** and **move toward the base case**.
- A recursive algorithm must **call itself**, recursively.

Fork this! https://replit.com/@ajlsfu/Fancy-Recursive-Tree

Let's review some concepts

```
def doer(n):
    if n == 0:
      pass
    else:
      print(n)
      doer(n-1)
```

```
def waiter(n):
    if n == 0:
      pass
    else:
      waiter(n-1)
      print(n)
```

What does **doer(3)** print and **waiter(3)** print?

Let's review some concepts

```
def doer(n):
    if n == 0:
      pass
    else:
      print(n)
      doer(n-1)
```

```
def doer(n):
    if n > 0:
      print(n)
      doer(n-1)
```

Are these the same?

Recursion reminder

Tip! Add print statements such as "entering function", "about to call recursively" to help trace these algorithms.

Recall the recursive function we wrote to print numbers on separate lines. What do you think these functions print?

```
def print_now(s):
    if len(s) > 0:
        print(s[0])
        print_now(s[1:])

print_now("abcde")
```

```
def print_later(s):
    if len(s) > 0:
        print_later(s[1:])
        print(s[0])

print_later("abcde")
```

Recursion

Recursion + Fruitful Functions

Fruitful recursion

This week, we'll go over some **classic recursion problems** that combine **fruitful functions** (that return a value) with **recursion**.

	Parameter type
Factorial	number
Sum of numbers in a list	list
String reversal	string

Fruitful recursion: Factorial

Write a **recursive function** that returns the **factorial** of a number.

$$n! = \begin{cases} 1 & \text{for } n = 0 \\ n \times (n-1)! & \text{for } n > 0 \end{cases}$$

```
# Calculates n! = n*(n-1)*(n-2)*...*1
def factorial(n):
    ...
```

What is the base case? How can you incorporate a call to itself with a parameter that moves it closer to the base case?

Fruitful recursion: Factorial

factorial(0) → 1
factorial(1) → 1*factorial(0) → 1*1 → 1
factorial(2) → 2*factorial(1) → 2*1

```
def factorial(num):
  # Base case
  if num == 0:
    return 1
  else:
    return num*factorial(num-1)

print(factorial(4))
```

What should this output?

Fruitful recursion: Sum

Task: Find the **sum** of the elements in the list. How would you do it iteratively (i.e. with a loop)?

Challenge: Can you write a **sum** function without using a loop?

http://interactivepython.org/runestone/static/thinkcspy/IntroRecursion/CalculatingtheSumofaListofNumbers.html

Fruitful recursion: Sum

Task: Find the sum of the elements in the list.

Challenge: Don't use **for** or **while**!

numList =

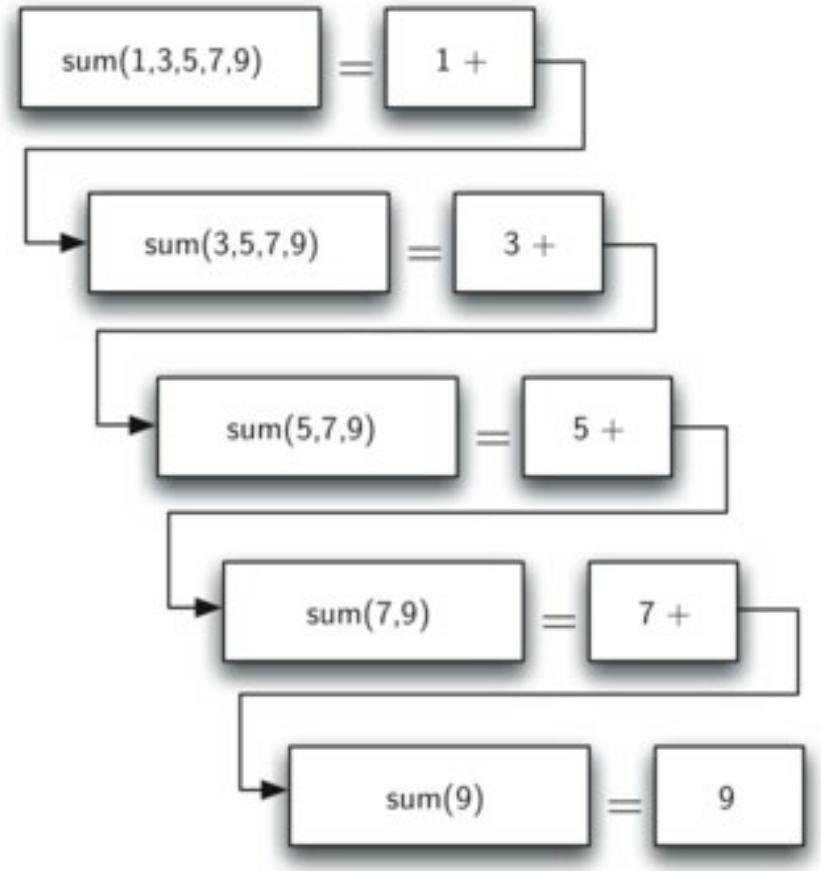
1	3	5	7	9

What's the subproblem?

$$listSum(numList) = first(numList) + listSum(rest(numList))$$

What's the base case?

http://interactivepython.org/runestone/static/thinkcspy/IntroRecursion/CalculatingtheSumofaListofNumbers.html


sum(1,3,5,7,9)
=
1 +
sum(3,5,7,9)
=
3 +
sum(5,7,9)
=
5 +
sum(7,9)
=
7 +
sum(9)
=
9

Let's code it!

main.py

```python
# More on Recursion
# Author: Angelica Lim
# Date: April 2, 2018

# http://interactivepython.org/runestone/static/thinkcspy/IntroRecursion/CalculatingtheSumofaListofNumbers.html

def listsum(numList):
    if len(numList) == 1:
        return numList[0]
    else:
        return numList[0] + listsum(numList[1:])

print(listsum([1,3,5,7,9]))
```

```
25
>
```

Recursion can be quite useful when the function returns something.

http://interactivepython.org/runestone/static/thinkcspy/IntroRecursion/CalculatingtheSumofaListofNumbers.html

Fruitful recursion: Reversing

Given a string, can you write a function that will return the reverse of the string?
E.g. “yellow” → “wolley”

Fruitful recursion: Reversing

```python
# Reversi
# Angelica Lim
# Nov. 15, 2021

def reverse(st):
    if len(st) == 0:
        return st
    else:
        return reverse(st[1:]) + st[0]

print(reverse("abcde"))
```

```
edcba
>
```

```python
# Reversi
# Angelica Lim
# Nov. 15, 2021

def reverse(st):
    if len(st) == 0:
        return st
    else:
        return st[-1] + reverse(st[:-1])

print(reverse("abcde"))
```

Another solution

Let's review

How can you identify if a problem is suited to be solved with recursion?

Do all parameters in a recursive function need to be used for controlling the end of recursion?

What are some classic algorithms that can be solved with fruitful recursion?