Recursion Revisited
The Rise of Google

Searching and Sorting

Announcement

- Bonus Project 3 Posted: Essay
- Please go to TA's office hours: this should be a norm, not an anomaly
- Quiz 3 posted: have 1 week to talk to the TA and request in writing regrading
- Class participation: 5%. TAs are keeping track…you are already here, why not actively participate?

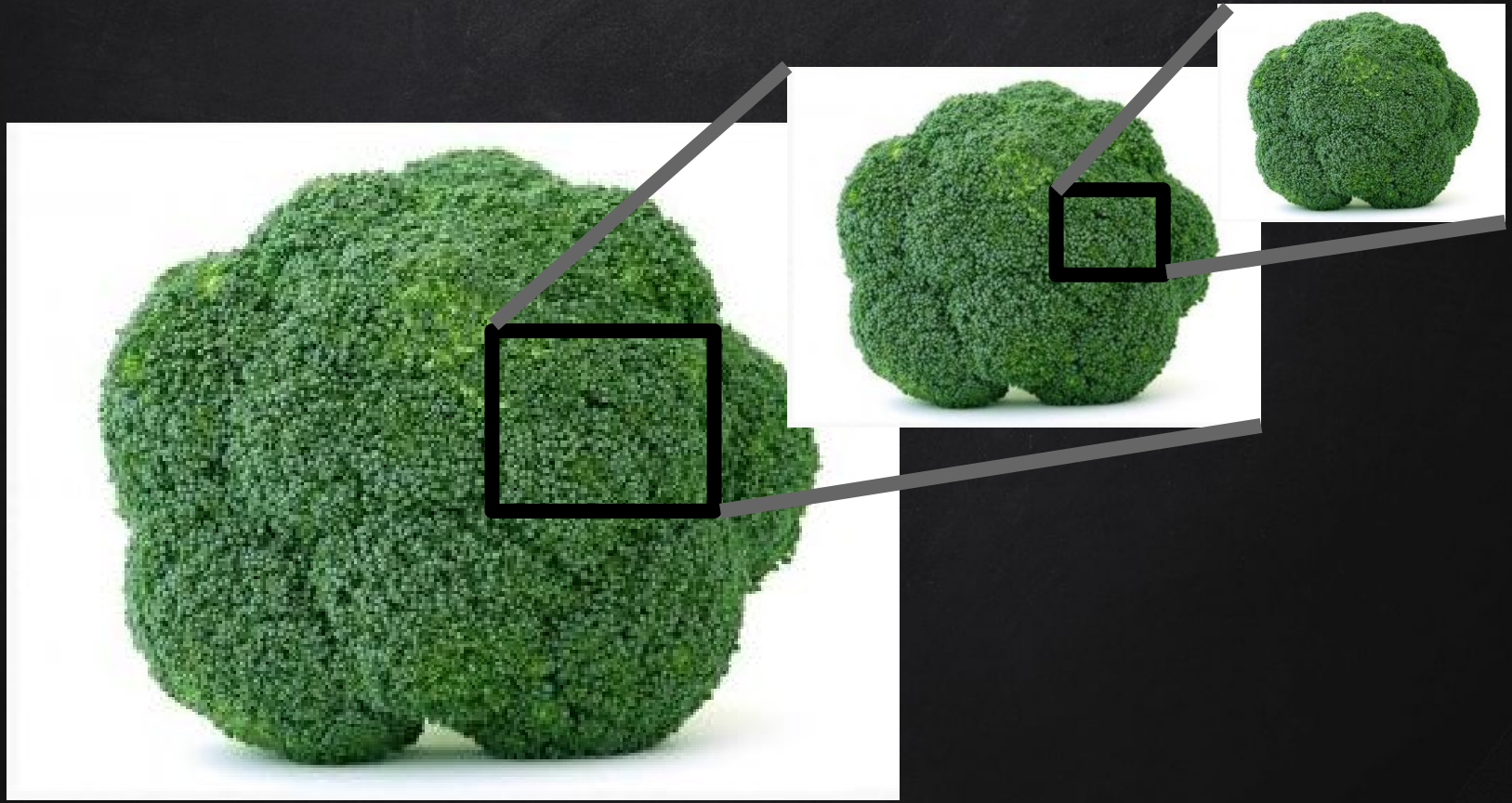
Recursion Revisited

http://www.pythontutor.com/index.html

What is Recursion?

A picture of a painter who is painting a picture of painter who is painting a picture ...

A Brocolli



What is a recursion?

- Joke: in order to understand recursion, you need to understand recursion…
- A recursion function is a function that calls itself
- Recursion is hard to understand…some people get it, some don't

Two things:

1. Understand how to solve a simpler problem
2. Understand how to trace a recursive function

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

The hate-love-hate relationship with Recursion

1. You hate it because you do not understand it
2. You love it because it is cool after you understand it
3. You hate it because it is typically inefficient

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Classic Recursion

Solving a "big" problem recursively means to solve one or more smaller versions of the problem, and using those solutions of the smaller problems to solve the "big" problem.

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Big-version v.s. Small-version of the same problem

- Solving problems recursively typically means that there are smaller versions of the problem solved in similar ways.
- Think about summing over an array of 10 numbers v.s. summing over an array of 5 numbers.
- Use the same technique: a counter
- Solution to the smaller problem helps you to solve the larger problem.

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Basic Unit of a recursion is a function call

Four steps to understand recursion

1. Write and define the prototype of the function
2. Write out a sample function call
3. Think of the smallest version of the problem
4. Think of smaller version of the function call

Putting it all together!

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Task: sum up numbers from 1 to x

1. Write and define the prototype of the function

```
def compute_sum(x):     # add numbers from 1 to x
```

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

2. Write out a sample function call

```
def compute_sum(x):     # add numbers from 1 to x
...

print compute_sum(1)
print compute_sum(2)
print compute_sum(3)
```

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

3. Think of the smallest version of the problem

Base case: the smallest version of the problem

Base case here: x = 1

Base case is where the recursion eventually stops

When x = 1, compute_sum(x) should return 1

```
def compute_sum(x):
    if x ==1:
        return 1
...
```

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

4. Think of smaller version of the function call

compute_sum(x) # compute sum from 1 to x

compute_sum(x-1) # compute sum from 1 to x-1

If we want to solve a bigger problem with solving a smaller problem first:

compute_sum(x) = x + compute_sum(x-1)

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Putting it all together!

```
def compute_sum(x):
    if x ==1:
        return 1
    else:
        return x + compute_sum(x-1)
```

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Putting it all together!

```
def a_simple_recursive_function(x):
    if (base case)
        return some simple expression
    else:
        some work before
        recursive call
        some work after
```

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

```
1  def f(x):
2   if x == 1:
3      return 1
4   else:
5      return (x + f(x-1))
6
7  print f(1)
8  print f(2)
9  print f(3)
10 print f(4)
11 print f(5)
```


1
3
6
10
15

Classic Recursion

Thinking "backwards":

Instead of building a solution from nothing, you pretend you are at the solution, and want to take a step back and ask how to solve the problem if you were a step back.

Alternatively, thinking about how the solution of a bigger problem can be constructed from a solution of a smaller problem.

compute_sum(x) = x + compute_sum(x-1)

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Another example

How to reverse a string?

reverse_string(“hello”) → “olleh”

reverse_string(“abcdefg”) → “gfedcba”

reverse_string(“abcdefg”): put “g” first, add it to the reversed result of “bcdefg”

reverse_string(str) = str[l-1] + reverse_string(str[0:l-1])

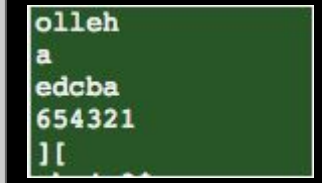
Another example

```
def a_simple_recursive_function(x):
    if (base case)
        return some simple expression
    else:
        some work before
        recursive call
        some work after
```

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

```python
def f(mystr):
 if len(mystr) == 1:
    return mystr
 else:
    l = len(mystr)
    return (mystr[l-1] + f(mystr[0:l-1]))

print f('hello')
print f('a')
print f('abcde')
print f('123456')
print f('[]')
```


olleh
a
edcba
654321
][

Final Example

Compute a*b with only additions/subtractions:

a*b = b + b + … b = a copies of b

1. multiply(a,b)
2. multiply(3,4)
3. multiply(1, b)
4. multiply(a,b)
    a. multiply(a–1, b)
    b. multiply(a,b) = b + multiply(a–1,b)

```
1  def multiply(a,b):
2   if a == 1:
3       
4   else:
5       
6
7  print multiply(2,4)
8  print multiply(9,100)
9  print multiply(1,10)
```

```
8
900
10
```

```
1  def multiply(a,b):
2   if a == 1:
3      return b
4   else:
5      return b + multiply(a-1, b)
6
7  print multiply(2,4)
8  print multiply(9,100)
9  print multiply(1,10)
```

More readings on Recursion

Tracing recursive functions:

http://www.pythontutor.com/index.html

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/trace-recursion.html

More readings:

https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion.html

Try tracing the following code in http://www.pythontutor.com/index.html
Using "Forward" step by step

```
def f(x):
  if x==1:
    return 1
  else:
    return x + f(x-1)

print f(3)
```

Searching Lists

Data Structures

An important part of computer science is data structure:

- A data structure is a particular way of storing data so it can be processed efficiently

For example: Storing numbers

A type of data structure: Lists

Imaging tracking the names of basketball players who scored in a basketball game:

Brekkott Chapman, Isaiah Wright, Austin Eastman, Jake Connor, Brandon Taylor, Dakarai Tucker, Lorenzo Bonam

Chris Reyes, Jordan Loveridgel, Kenneth Ogbe, Gabe Bealer, Jayce Johnson, Kyle Kuzma, Makol Mawien, Austin Montgomery

Jakob Poeltl

- Need to store that information in a variable
- Make a new variable for each scorer: scorer1, scorer2, …
- Use as needed as the game progresses


INGRAM
14
5

Lists

A list is a collection of information

- Variable length
- Can add or remove item from the list
- Can look at items in a list

What are some examples of lists in real-life?

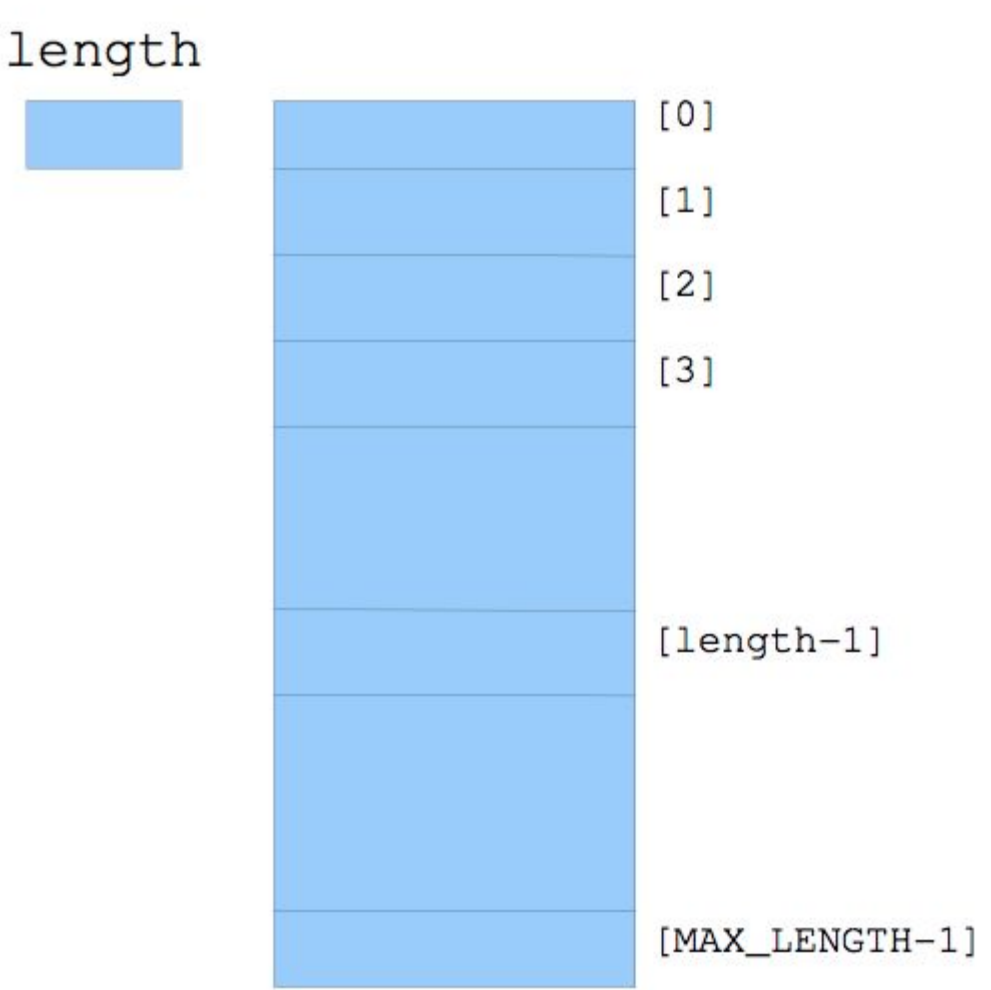
Lists in real life

Shopping list:

shopping_list = [‘apple’, ‘orange’, ‘meat’, ‘napkin’]

Lists on a Computer

- You can make lists different ways on a computer
- One way is to use an array: in Python-- it is a list
- Each individual item is accessed by its place in the collection
- We call the location number for a value an index



More Lists in Python

```
shopping_list = ['apple', 'orange', 'meat', 'napkin']
course_list = ['physics', 'chemistry','computer science']
number_list = [1, 2, 3, 4, 5]
alphabet_list =['a', 'b','c','d']
```

Review: Lists

Lists

- Contain multiple items
- Expands to hold as many items as needed
- Look at a particular item with an index number
- Add, remove, replace items in a list

```
course_list = ['physics', 'chemistry', 'english', 'biology'];
number_list = [1, 2, 3, 4, 5, 6, 7 ];

print "course_list[0]: ", course_list[0]
print "number_list[1:5]: ", number_list[1:5]
```

course_list[0]: physics
number_list[1:5]: [2, 3, 4, 5]

```
course_list = ['physics', 'chemistry', 'english', 'biology'];

print "Value at index 2 : "
print course_list[2]
course_list[2] = 2001;
print "New value at index 2 : "
print course_list[2]
print course_list
```

```
Value at index 2 :
english
New value at index 2 :
2001
['physics', 'chemistry', 2001, 'biology']
```

```
course_list = ['physics', 'chemistry', 'english', 'biology'];

print "Value at index 2 : "
print course_list[2]

print course_list
del course_list[2]
print "After deleting value at index 2 : "
print course_list
```

```
Value at index 2 :
english
['physics', 'chemistry', 'english', 'biology']
After deleting value at index 2 :
['physics', 'chemistry', 'biology']
```

```
course_list = ['physics', 'chemistry', 'english', 'biology'];

print course_list[2]
print course_list[-1]
print course_list[1:]
```

```
english
biology
['chemistry', 'english', 'biology']
```

```
course_list = ['physics', 'chemistry', 'english', 'biology'];

print len(course_list)
print [1,2,3]+[4,5,6]
print 'hello'*4
print 3 in [1,2,3]
for x in course_list:
    print x
```

4
[1, 2, 3, 4, 5, 6]
hellohellohellohello
True
physics
chemistry
english
biology

Coming Up Next: More on Lists and Sorting

THANKS!

Any questions?

You can find me at
beiwang@sci.utah.edu

http://www.sci.utah.edu/~beiwang/teaching/cs1060.html

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash