Recursion
Announcements

- Maps is out
  - Checkpoint 1 is due tonight. It’s a few short functions
  - No slip days for the check point, but slip days for the rest of the project.
- Midterm:
  - Thursday 3/11
  - Includes recursion (today and Friday)
Computing In The News

- AI Can Write a Passing College Paper in 20 Minutes
  ZDNet Greg Nichols February 24, 2021

Researchers at Education Reference Desk (EduRef), a resource for current and prospective students, found that an artificial intelligence (AI) tool can write a college term paper in three to 20 minutes and achieve a passing grade. Humans, in contrast, took three days on average to complete the same assignment. The researchers had a panel of professors grade anonymous submissions to writing prompts from recent graduates and undergraduate-level writers and Open AI’s GPT-3, a deep learning language prediction model. The professors gave GPT-3 an average grade of “C” in four subjects, and it failed just one assignment. Said the researchers, “Even without being augmented by human interference, GPT-3’s assignments received more or less the same feedback as the human writers.”
Recursion
Why Recursion?

- Recursive structures exist (sometimes hidden) in nature and therefore in data!
- It’s mentally and sometimes computationally more efficient to process recursive structures using recursion.
- Sometimes, the recursive definition is easier to understand or write, even if it is computationally slower.
Today: Recursion

**re·cur·sion**

*noun*  
MATHEMATICS  LINGUISTICS

the repeated application of a recursive procedure or definition.
- a recursive definition.
plural noun: recursions

**re·cur·sive**

*adjective*

characterized by recurrence or repetition, in particular.
- relating to or involving the repeated application of a rule, definition, or procedure to successive results.
- relating to or involving a program or routine of which a part requires the application of the whole, so that its explicit interpretation requires in general many successive executions.

- Recursive function calls itself, directly or indirectly
Demo: Vee

• run 11-recursion.py
• The file will open an interpreter.
• Use the following keys to play with the demo
  – Space to draw
  – C to Clear
  – Up to add “vee” to the functions list
  – Down to remove the “vee” functions from the list.
Demo: Countdown

def countdown(n):
    if n == 0:
        print('Blastoff!')
    else:
        print(n)
        countdown(n - 1)
Recursive solutions involve two major parts:

- **Base case(s)**, the problem is simple enough to be solved directly
- **Recursive case(s)**. A recursive case has three components:
  - **Divide** the problem into one or more simpler or smaller parts
  - **Invoke** the function (recursively) on each part, and
  - **Combine** the solutions of the parts into a solution for the problem.
Computational Structures in Data Science

Recursion
Learning Objectives

• Compare Recursion and Iteration to each other
  – Translate some simple functions from one method to another
• Write a recursive function
  – Understand the base case and a recursive case
For loop:

```python
def sum(n):
    s=0
    for i in range(0,n+1):
        s=s+i
    return s
```
While loop:

```python
def sum(n):
    s=0
    i=0
    while i<n:
        i=i+1
        s=s+i
    return s
```
Recursion:

def sum(n):
    if n == 0:
        return 0
    return n+sum(n-1)
Iteration vs Recursion: Cheating!

Sometimes it’s best to just use a formula! But that’s not always the point. 😊

```python
def sum(n):
    return (n * (n + 1)) / 2
```
The Recursive Process

- Recursive solutions involve two major parts:
  - **Base case(s)**, the problem is simple enough to be solved directly
  - **Recursive case(s)**. A recursive case has three components:
    - **Divide** the problem into one or more simpler or smaller parts
    - **Invoke** the function (recursively) on each part, and
    - **Combine** the solutions of the parts into a solution for the problem.
Recall: Iteration

```python
def sum_of_squares(n):
    accum = 0
    for i in range(1, n+1):
        accum = accum + i*i
    return accum
```

1. Initialize the “base” case of no iterations
2. Starting value
3. Ending value
4. New loop variable value
Recursion Key concepts – by example

```python
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```

1. Test for simple “base” case
2. Solution in simple “base” case
3. Assume recursive solution to simpler problem
4. "Combine" the simpler part of the solution, with the recursive case
In words

- The sum of no numbers is zero
- The sum of $1^2$ through $n^2$ is the
  - sum of $1^2$ through $(n-1)^2$
  - plus $n^2$

```python
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```
Why does it work

```
sum_of_squares(3)

# sum_of_squares(3) => sum_of_squares(2) + 3**2
# => sum_of_squares(1) + 2**2 + 3**2
# => sum_of_squares(0) + 1**2 + 2**2 + 3**2
# => 0 + 1**2 + 2**2 + 3**2 = 14
```
Review: Functions

• Generalizes an expression or set of statements to apply to lots of instances of the problem
• A function should do one thing well

```python
def concat(str1, str2):
    return str1 + str2;

concat(“Hello”,”World”)
```
How does it work?

• Each recursive call gets its own local variables
  - Just like any other function call

• Computes its result (possibly using additional calls)
  - Just like any other function call

• Returns its result and returns control to its caller
  - Just like any other function call

• The function that is called happens to be itself
  - Called on a simpler problem
  - Eventually stops on the simple base case
Questions

• In what order do we sum the squares?
• How does this compare to iterative approach?

```python
def sum_of_squares(n):
    accum = 0
    for i in range(1, n+1):
        accum = accum + i*i
    return accum

def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2

def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return n**2 + sum_of_squares(n-1)
```
Trust ...

- The recursive “leap of faith” works as long as we hit the base case eventually

What happens if we don’t?
Why Recursion?

• “After Abstraction, Recursion is probably the 2^{nd} biggest idea in this course”
• “It’s tremendously useful when the problem is self-similar”
• “It’s no more powerful than iteration, but often leads to more concise & better code”
• “It’s more ‘mathematical’”
• “It embodies the beauty and joy of computing”
• ...
Recursion (unwanted)
Example I

List all items on your hard disk

- Files
- Folders contain
  - Files
  - Folders

Recursion!
Another Example

```python
def first(s):
    """Return the first element in a sequence."""
    return s[0]

def rest(s):
    """Return all elements in a sequence after the first"""
    return s[1:]

def min_r(s):
    """Return minimum value in a sequence."""
    if len(s) == 1:
        return first(s)
    else:
        return min(first(s), min_r(rest(s))
```

- Recursion over sequence length, rather than number magnitude

- Indexing an element of a sequence
- Slicing a sequence of elements

Base Case

Recursive Case
Why Recursion? More Reasons

- Recursive structures exist (sometimes hidden) in nature and therefore in data!
- It’s mentally and sometimes computationally more efficient to process recursive structures using recursion.