CompSci 101
Fall 2021

Lecture 21

Reminders

- Assignments
  - Assign 6 due TODAY
  - Assign 7 and APT 8 due Thursday (12/2)
  - GRACE PERIOD ENDS DEC 3! NOT ACCEPTED AFTER THIS!
- Lab 11
  - Pre-lab this week
- Assessment sent via Learning Innovation
  - 80% response rate by 11/30→Extra credit
- Ethics in AI talk

- Spring 2022 UTA applications open
  - See Ed announcement

Final Exam

- APT quiz style
  - Part A and B
    - 90 minutes each
  - Official schedule (12/10-9am-12pm)
    - May complete anytime between 12/10 (8am EST) and 12/12 (11pm EST)
  - Same rules apply of what can/cannot be used
    - Do not violate academic code of conduct!

Key instructions

- Input ✔
- Output ✔
- Assignments* ✔
- Math/Logic ✔
- Conditionals ✔
- Repetition ✔

**not listed in book*

Python Data Types

- int, float, bool ✔
- Collections
  - Strings ✔
  - Lists ✔
  - Tuples ✔
  - Sets ✔
  - Dictionaries ✔

PFTD

- How do Dictionaries work so fast!
  - Access an element in constant time
- Recursion
  - Solving a problem by solving smaller problems

KISS Principle

- Think of the non-computing context for any word/terms
- KISS model
  - Work smarter, not harder!!
- “Good programmers are simply good designers.”
  - *-Dr. Washington*
- Design first and always!
- Importance of reusability
- *USE PyCharm/PythonTutor IF YOU HAVE QUESTIONS!*

People to Know: Michael Running Wolf

- BS/MS (Montana State University)
- Clinical Instructor (CS)
  - Northeastern University (Vancouver)
- Founder, Indigenous in AI
- IBM, Lawrence Livermore National Laboratory, Amazon
- Indigenous automatic speech recognition
- Published poet
- Northern Cheyenne

Assignment 7: Create, Due 12/2

Grace period til 12/3, No late days!

Must be turned in by 12/3

This assignment is required!

Pick one:

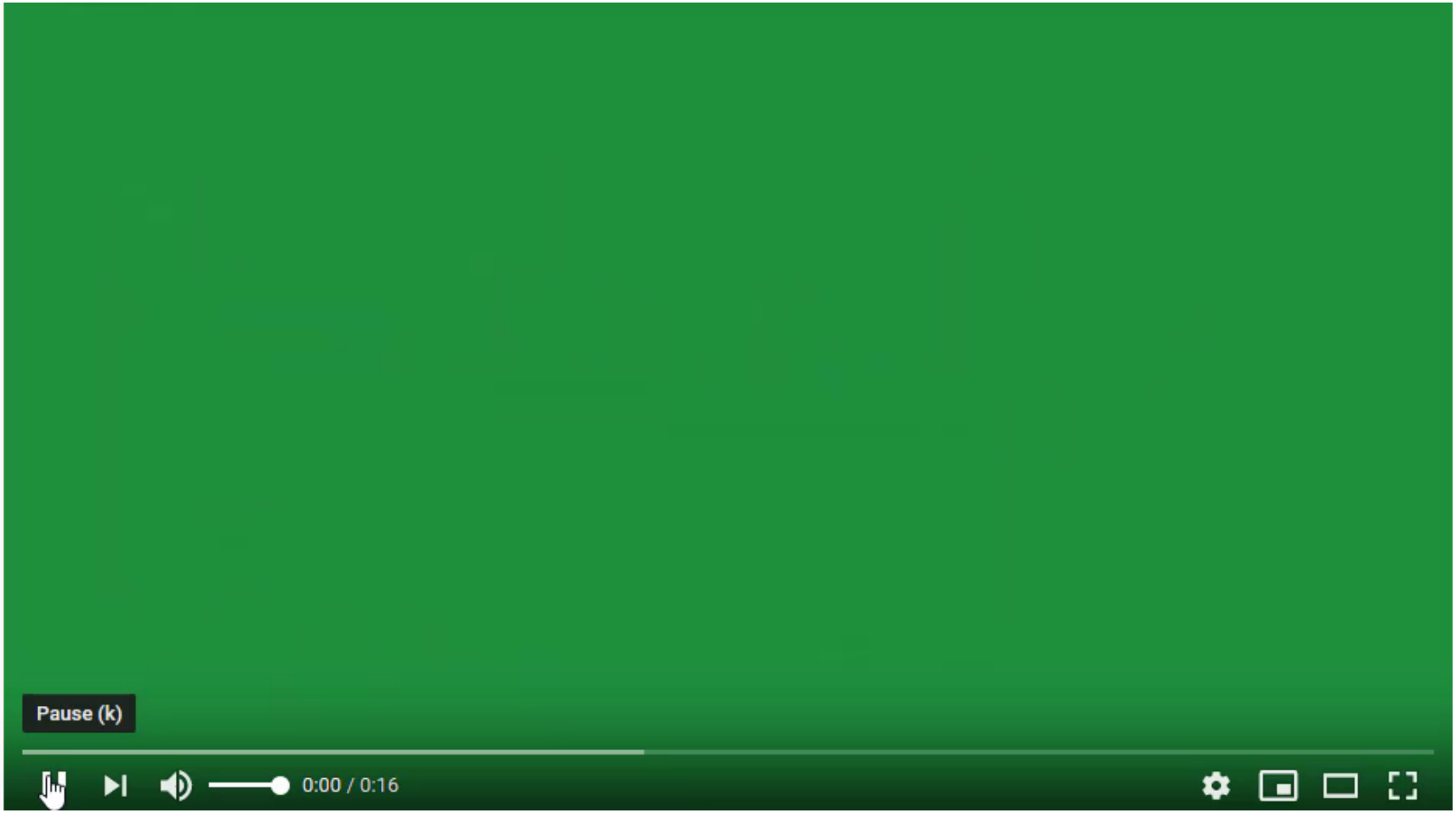
- Video: Green dance, advertisement for 101, song, other
- Poem or Multiple Haikus
- Story
- Comic
- One-pager
- Feedback

Let's see some examples

Video Song


Help hours!
That's the trick
we were everyday.
We even went to class we
went all the way!
they taught us and explained
to us what we gotta think like.
All green solutions? This
might be alright!
0:00 / 2:15

Video Simple Green Dance



Video: APT Success

How do Dictionaries work so fast?

- How are they implemented?

Simple Example

Want a mapping of Soc Sec Num to Names

- Duke's CS Student Union wants to be able to quickly find out info about its members. Also add, delete and update members. Doesn't need members sorted.

  267-89-5431 John Smith

  703-25-6141 Ademola Olayinka

  319-86-2115 Betty Harris

  476-82-5120 Rose Black

- Dictionary d – SSN to names
  - d['267-89-5431'] = 'John Smith'
  - How does it find 'John Smith' so fast?

Dictionary d(SSN) = (SSN, name)

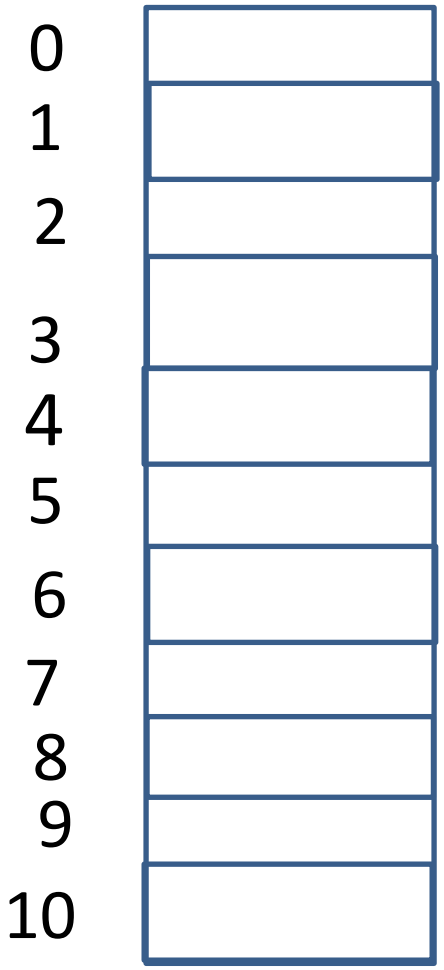
- We actually would map the SSN to the tuple of (SSN, name).
- That is a lot to display on a slide, so we will just show SSN to name
- But remember name is really a tuple of (SSN,name)

Simple Example
Let's look under the hood.

- Dictionaries implemented with a list, in a clever way
- How do we put something into the list fast?
- How do we find it in the list quickly?
  - d['267-89-5431'] = 'John Smith'
- List size is 11 – from 0 to 10
- d['267-89-5431'] calculates index location in list of where to put this tuple (SSN,name)
- Use a function to calculate where to store 'John Smith
  - H(ssn) = (last 2 digits of ssn) mod 11
  - Called a Hash function

Have a list of size 11 from 0 to 10

- Insert these into the list
- Insert as (key, value) tuple
  (267-89-5431, John Smith)
  (in example, only showing name)



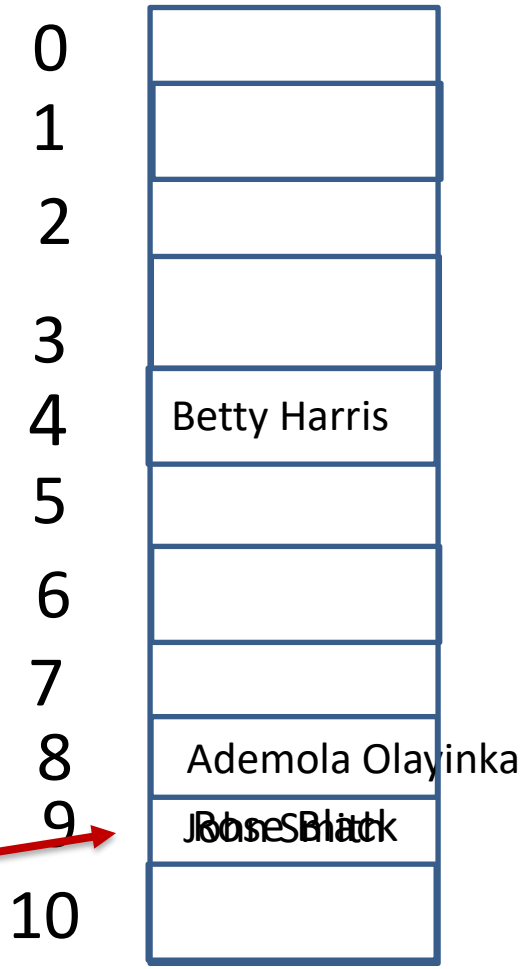
Have a list of size 11 from 0 to 10

- Insert these into the list
- Insert as (key, value) tuple

(267-89-5431, John Smith)
(in example, only showing name)

H(267-89-5431) = 31 %11 = 9
John Smith
H(703-25-6141) = 41%11 = 8
Ademola Olayinka
H(319-86-2115)= 15 %11 = 4
Betty Harris
H(476-82-5120) = 20%11 = 9
Rose Black

Collision!



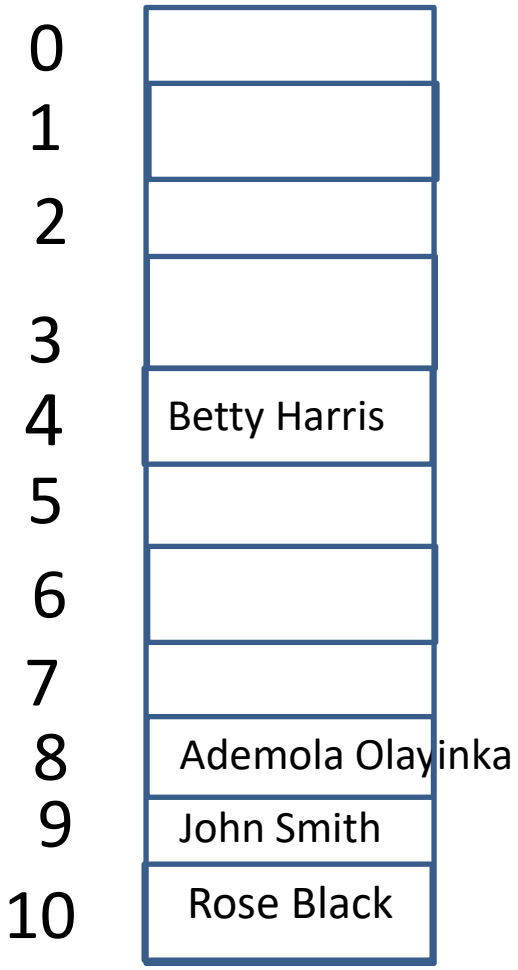
Have a list of size 11 from 0 to 10

- Insert these into the list
- Insert as (key, value) tuple

(267-89-5431, John Smith)
(in example, only showing name)

H(267-89-5431) = 31 %11 = 9
John Smith
H(703-25-6141) = 41%11 = 8
Ademola Olayinka
H(319-86-2115)= 15 %11 = 4
Betty Harris
H(476-82-5120) = 20%11 = 9
Rose Black

Must resolve collisions



When does this work well?

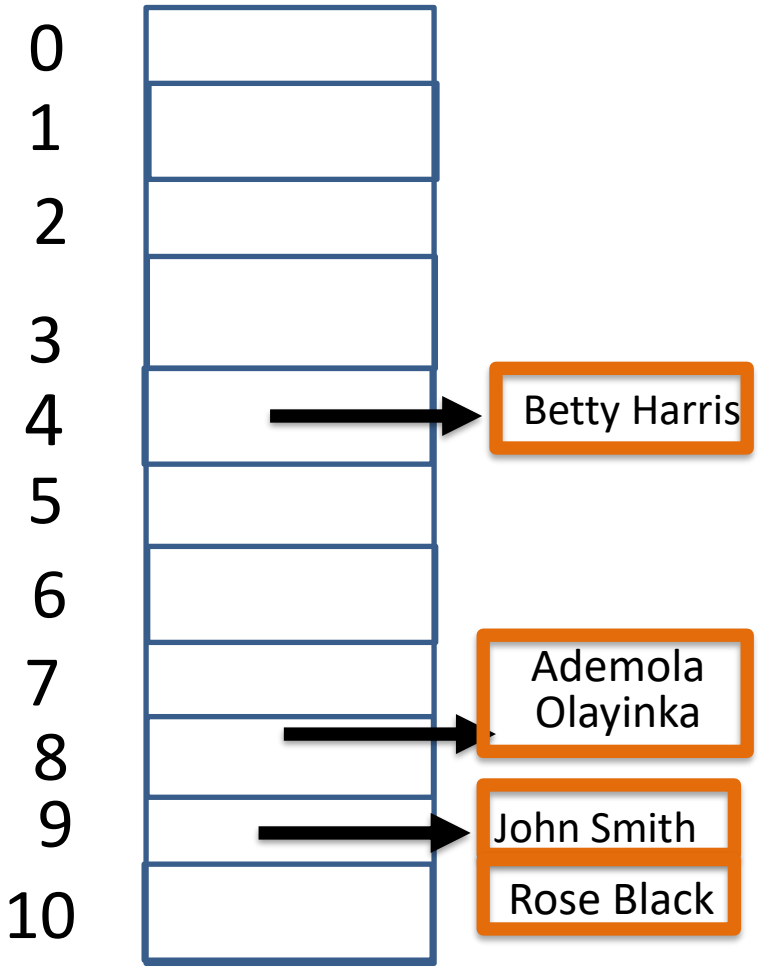
- When there are few collisions
- You must address collisions
- Use a list large enough to spread out your data

Another way: Use a list of lists

- Insert these into the list
- Insert as (key, value) tuple

(267-89-5431, John Smith)
(in example, only showing name)

H(267-89-5431) = 31 %11 = 9
John Smith
H(703-25-6141) = 41%11 = 8
Ademola Olayinka
H(319-86-2115)= 15 %11 = 4
Betty Harris
H(476-82-5120) = 20%11 = 9
Rose Black



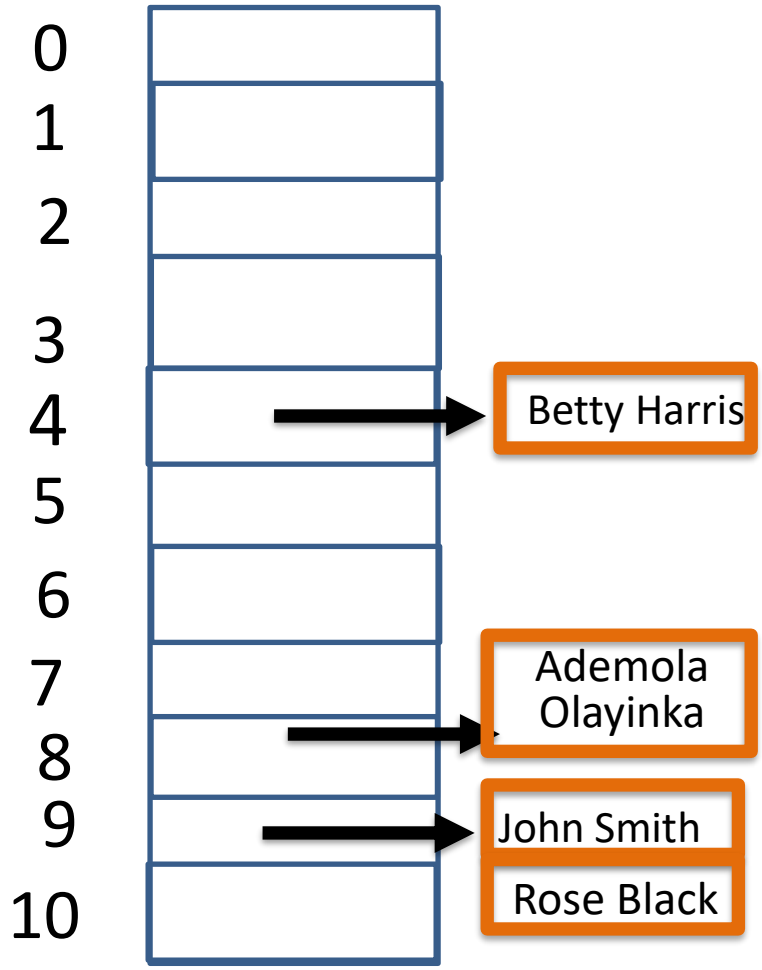
Another way: Use a list of lists

- Insert these into the list
- Insert as (key, value) tuple

(267-89-5431, John Smith)
(in example, only showing name)

H(267-89-5431) = 31 %11 = 9
John Smith
H(703-25-6141) = 41%11 = 8
Ademola Olayinka
H(319-86-2115)= 15 %11 = 4
Betty Harris
H(476-82-5120) = 20%11 = 9
Rose Black

Collisions added to list, 2 in list 9



WOTO-1 How Dictionaries Work
https://bit.ly/101f21-11-30-1

What Is Recursion?

Recursion: A programming technique in which a function calls itself to divide work into smaller portions

- **Recursive call:** When the function being called is the same as the one making the call

Each successive call in a recursion gets closer to a solution.

Must work way backwards from solution to solve each call→original recursive call

The Classic Example

- The factorial function, n!, is a classic example of recursion in mathematics
- 4! = 4 * 3 * 2 * 1 = 24
- The **recursive definition** is:
  - n! = 1 if n = 0
  - n! = n * (n-1)! if n > 0
  - **4!** = 4 * **3!** = 4 * 3 * **2!** = 4 * 3 * 2 * **1!** =
  - 4 * 3 * 2 * 1 * **0!** = 4 * 3 * 2 * 1 * 1 = **24**

Recursion Terms

- **Recursive definition:** When something is defined in terms of a smaller version of itself

Every recursion requires:

- **Base case (stopping point):** The case for which the solution can be defined non-recursively (n! = 1 if n = 0)
- **General (or recursive) case:** A case that is defined using recursion (n! = n * (n-1)! if n > 0)

Factorial Code Example

Iterative Solution

```
7   def Factorial(number):
8       result=1
9       for count in range(2,number+1):
10          result*=count
11      return result
12
13  if __name__ == '__main__':
14      print(Factorial(2))
```

Recursive solution

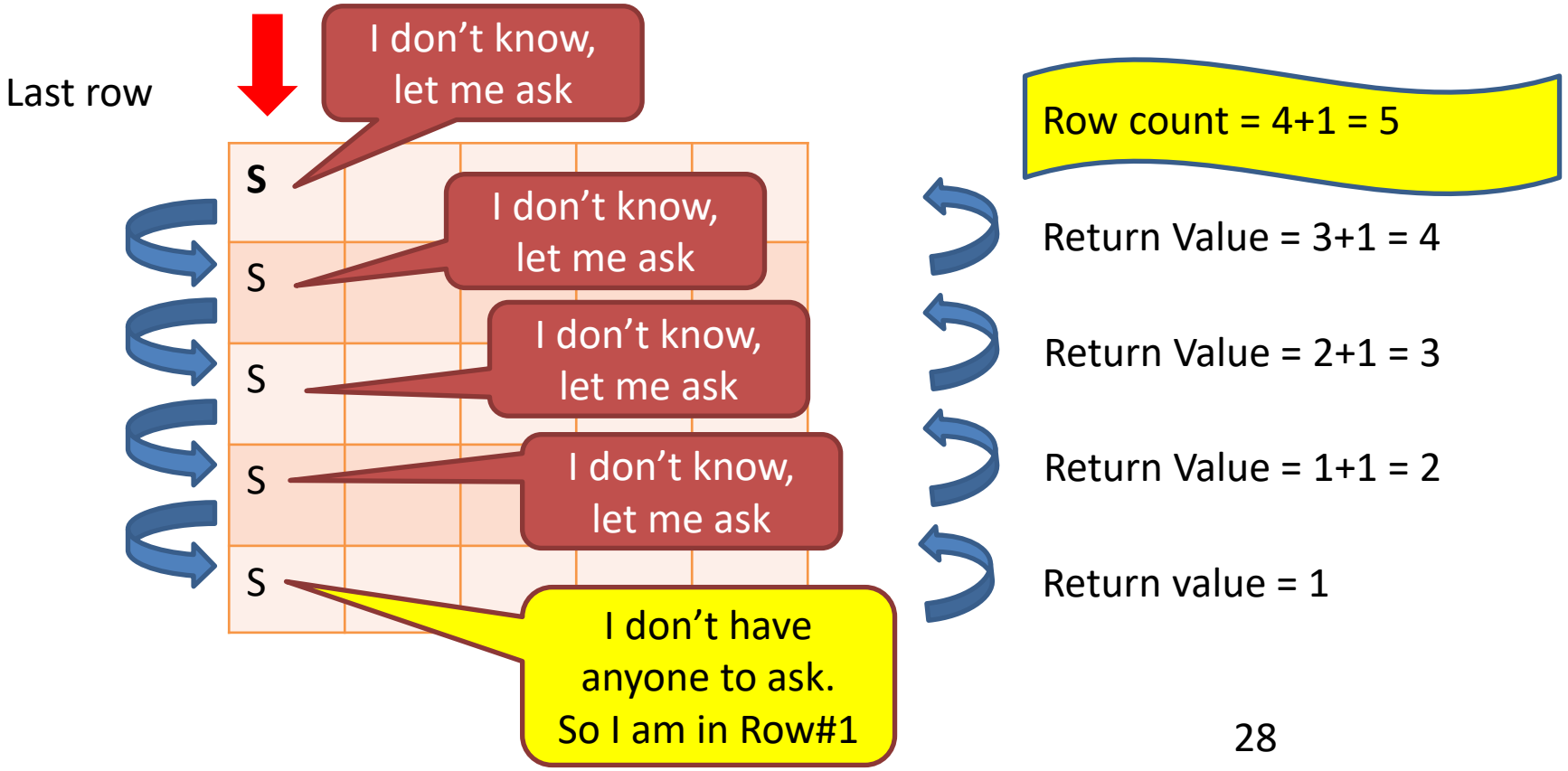
```
7   def Factorial(number):
8       if number == 0:      #Base case
9           return 1
10      else:                #General case
11          return number * Factorial(number-1)
12
13  if __name__ == '__main__':
14      print(Factorial(2))
```

Recursion

Solving a problem by solving similar but smaller problems

Question - How many **rows** are there in this classroom?

Similar but smaller question - How many **rows** are there until your row?



Recursion Summary

- Make Simpler or smaller calls
  - Call a clone of itself with different input
- Must have a base case when no recursive call can be made
    - Example (n=0→Factorial function)
    - Example (first row→classroom)

Mystery Recursion
https://bit.ly/101f21-11-30-2

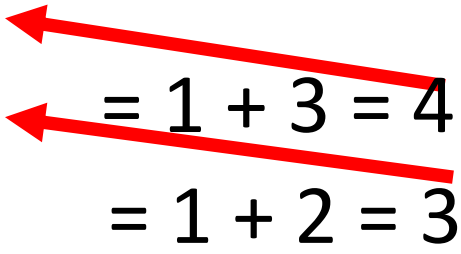
Mystery Recursion

```
def Mystery(num):
    if num > 0:
        return 1 + Mystery(num//2)
    else:
        return 2 + num
```

Example

```
def Mystery(num):
    if num > 0:
        return 1 + Mystery(num//2)
    else:
        return 2 + num
```

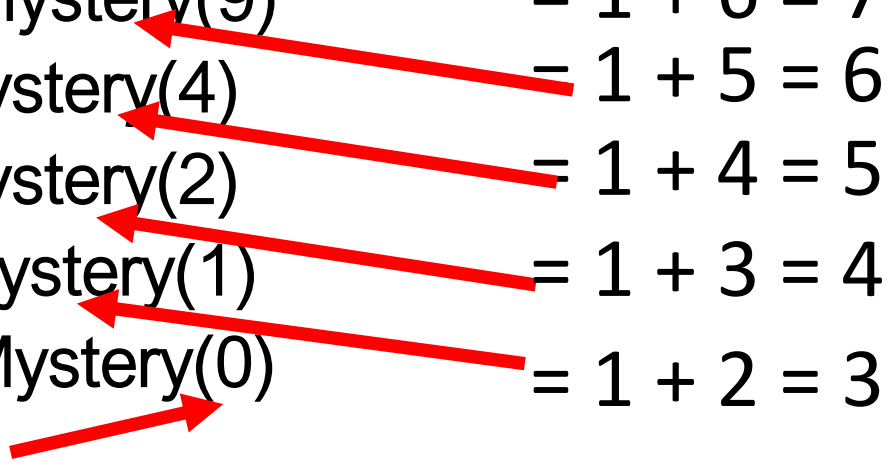
- Mystery(4) is 1 + Mystery(2)
- Mystery(2) is 1 + Mystery(1)
- Mystery(1) is 1 + Mystery(0)
- Mystery(0) is 2



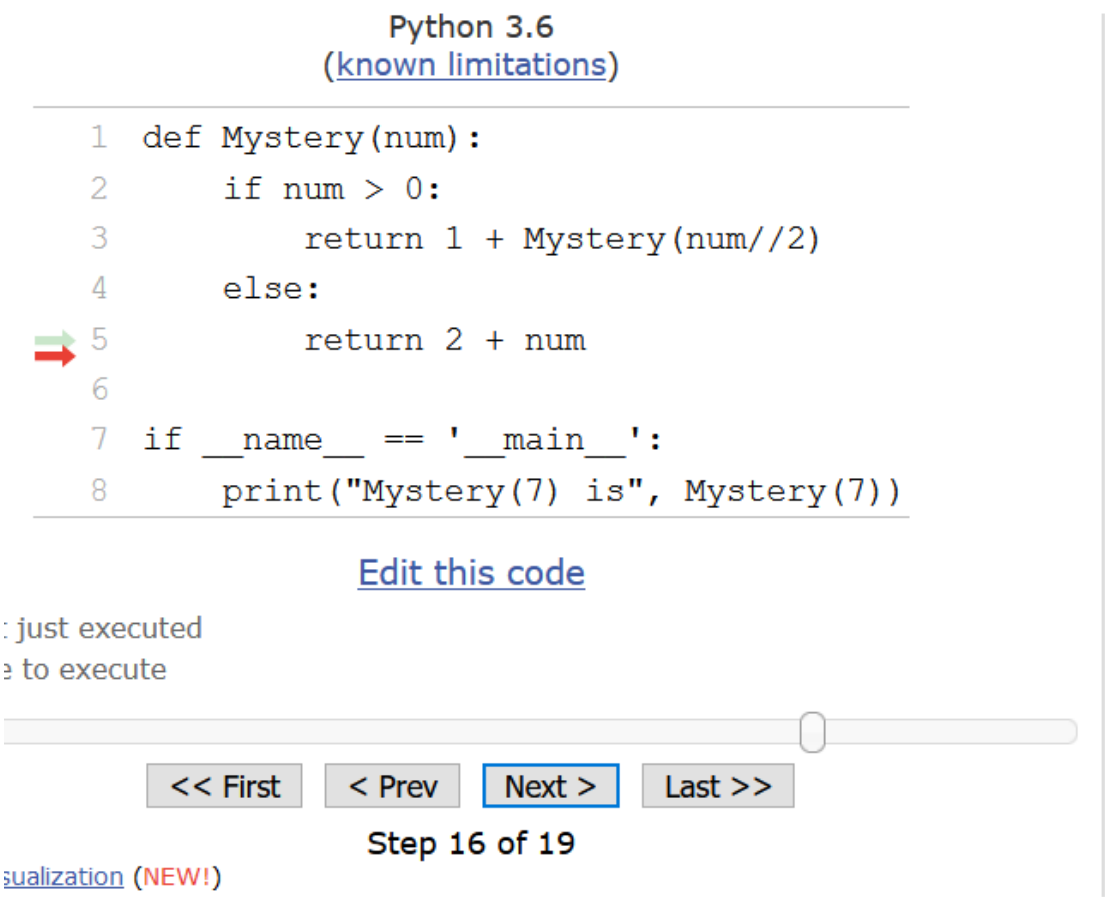
Example

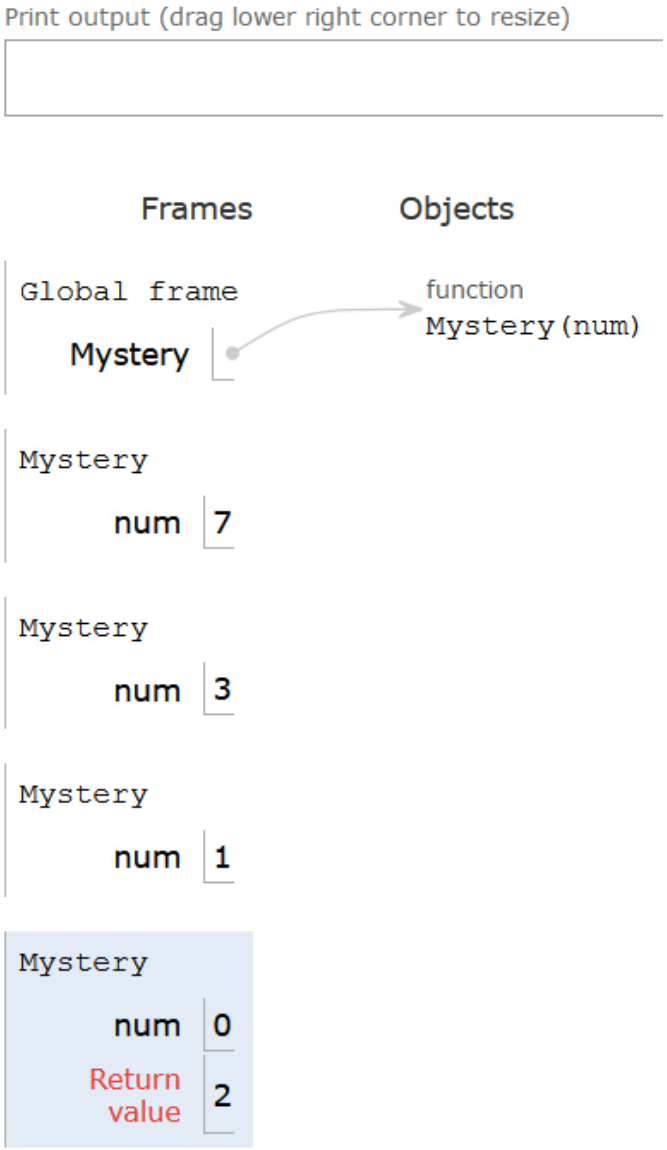
```
def Mystery(num):
    if num > 0:
        return 1 + Mystery(num//2)
    else:
        return 2 + num
```

- Mystery(18) is 1 + Mystery(9) = 1 + 6 = 7
- Mystery(9) is 1 + Mystery(4) = 1 + 5 = 6
- Mystery(4) is 1 + Mystery(2) = 1 + 4 = 5
- Mystery(2) is 1 + Mystery(1) = 1 + 3 = 4
- Mystery(1) is 1 + Mystery(0) = 1 + 2 = 3
- Mystery(0) is 2 + 0



Mystery in Python Tutor





Reminders

- Work smarter, not harder
- Design first
- Get smaller parts working, then build on it
- Try to identify where you are stuck
  - Identify resources to help solve problem
- Leverage your design and PythonTutor to understand program flow of control
  - http://pythontutor.com