W200 Functions

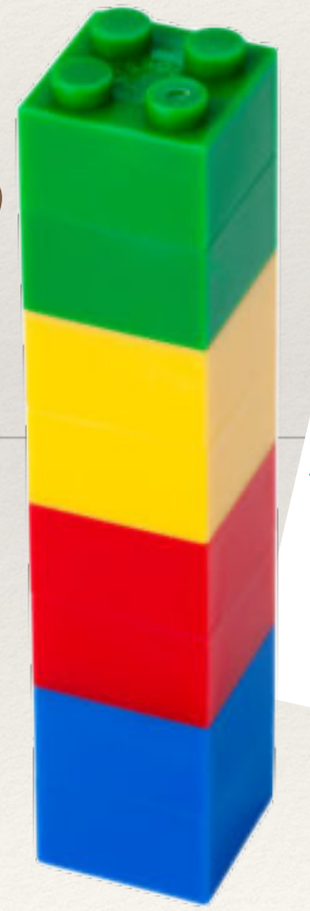
Week 5

USER-DEFINED

BUILT-IN

LAMBDA

RECURSIVE



```
# Python program to demonstrate
# difference between pass and
# continue statements
s = "geeks"

# Pass statement
for i in s:
    if i == 'k':
        print('Pass executed')
        pass
    print(i)

print()

# Continue statement
for i in s:
    if i == 'k':
        print('Continue executed')
        continue
    print(i)
```

Week #	Tue	Wed	Thu	Sat	Unit	Description		Start	Due	Start	Due	Start	Due	Notes
1	7-Jan	8-Jan	9-Jan	11-Jan	Unit 1	Introduction, Command Line, & Source Control		HW unit 1						
2	14-Jan	15-Jan	16-Jan	18-Jan	Unit 2	Starting Out with Python		HW unit 2	HW unit 1					Sat 18 Jan - Official last day to a
3	21-Jan	22-Jan	23-Jan	25-Jan	Unit 3	Sequence Types and Dictionaries		HW unit 3	HW unit 2					
4	28-Jan	29-Jan	30-Jan	1-Feb	Unit 4	More About Control and Algorithms		HW unit 4	HW unit 3					
5	4-Feb	5-Feb	6-Feb	8-Feb	Unit 5	Functions		HW unit 5	HW unit 4					
6	11-Feb	12-Feb	13-Feb	15-Feb	Unit 6	Complexity		HW unit 6	HW unit 5			Project 1		
7	18-Feb	19-Feb	20-Feb	22-Feb	Unit 7	Classes		HW unit 7	HW unit 6					
8	25-Feb	26-Feb	27-Feb	29-Feb	Unit 8	Object-Oriented Programming			HW unit 7	Exam 1			Project 1 Proposal	
9	3-Mar	4-Mar	5-Mar	7-Mar	Unit 9	Text and Binary Data					Exam 1			
10	10-Mar	11-Mar	12-Mar	14-Mar	Unit 10	NumPy - Vectors	Project 1 Presentation	HW unit 9					Project 1 Code	
11	17-Mar	18-Mar	19-Mar	21-Mar	Unit 11	Pandas - Dataframes		HW unit 10	HW unit 9			Project 2		
	24-Mar	25-Mar	26-Mar	28-Mar		Spring Break - no classes!								
12	31-Mar	1-Apr	2-Apr	4-Apr	Unit 12	MatPlotLib - Data Visualization		HW on units 11-13	HW unit 10				Project 2 Proposal	
13	7-Apr	8-Apr	9-Apr	11-Apr	Unit 13	Advanced Pandas - Aggregation & Groups			HW units 11-13	Exam 2				
14	14-Apr	15-Apr	16-Apr	18-Apr	Unit 14	Testing	Project 2 Presentation				Exam 2		Project 2 Report	Mon Apr 20th - Official last day 11:59 PM PST Mon Apr 20th - E

https://docs.google.com/spreadsheets/d/1PYz286UPN0sCRsRII5YEGwU5b0w6dZpDEuhjrSwef7M/edit#gid=0

Homework assignments are due 11:59pm PST the day before class

Project presentations are due in class meeting. Project code and report are due 11:59pm the night AFTER class meeting.

All due dates are tentative and may be changed by instructors with reasonable advance notice.

LINK

Week of … Adjust to your class's date.

Feb 3: Functions
Feb 10: Complexity
Start Project 1
Feb 17: Classes
Feb 24: OOP
Exam 1 start week
Project 1 Proposal

Mar 2: Text & Binary Data
Mar 9: NumPy, Vectors
Project 1 Presentations & Code
Mar 16: Pandas, DataFrames
Start Project 2
Mar 23-28: Spring Break!

Mar 30: Data Vis
Project 2 Proposal
Apr 6: Adv. Pandas
Start of week for Exam 2
Apr 13: Presentation
Project 2 Presentation; Report; Exam 2

Spring 2020

Today's Agenda . . .

UC Berkeley
SCHOOL OF INFORMATION

- Review
  - Discuss Homework 4
  - Discuss Name Space Concept
- Anatomy of a Function
- Functions as Objects
- map() & lambda
- Default, Positional, & System Arguments
- Recursion
- Exception Handling
- Stack Trace

Discuss Homework 4

UC Berkeley
SCHOOL OF INFORMATION

Homework 4:

- for loop
- chess
- algorithms - binary search (discussion & updates)
- comprehensions

What was the hardest part of homework 4?
How much time did you spend on this week's assignment?

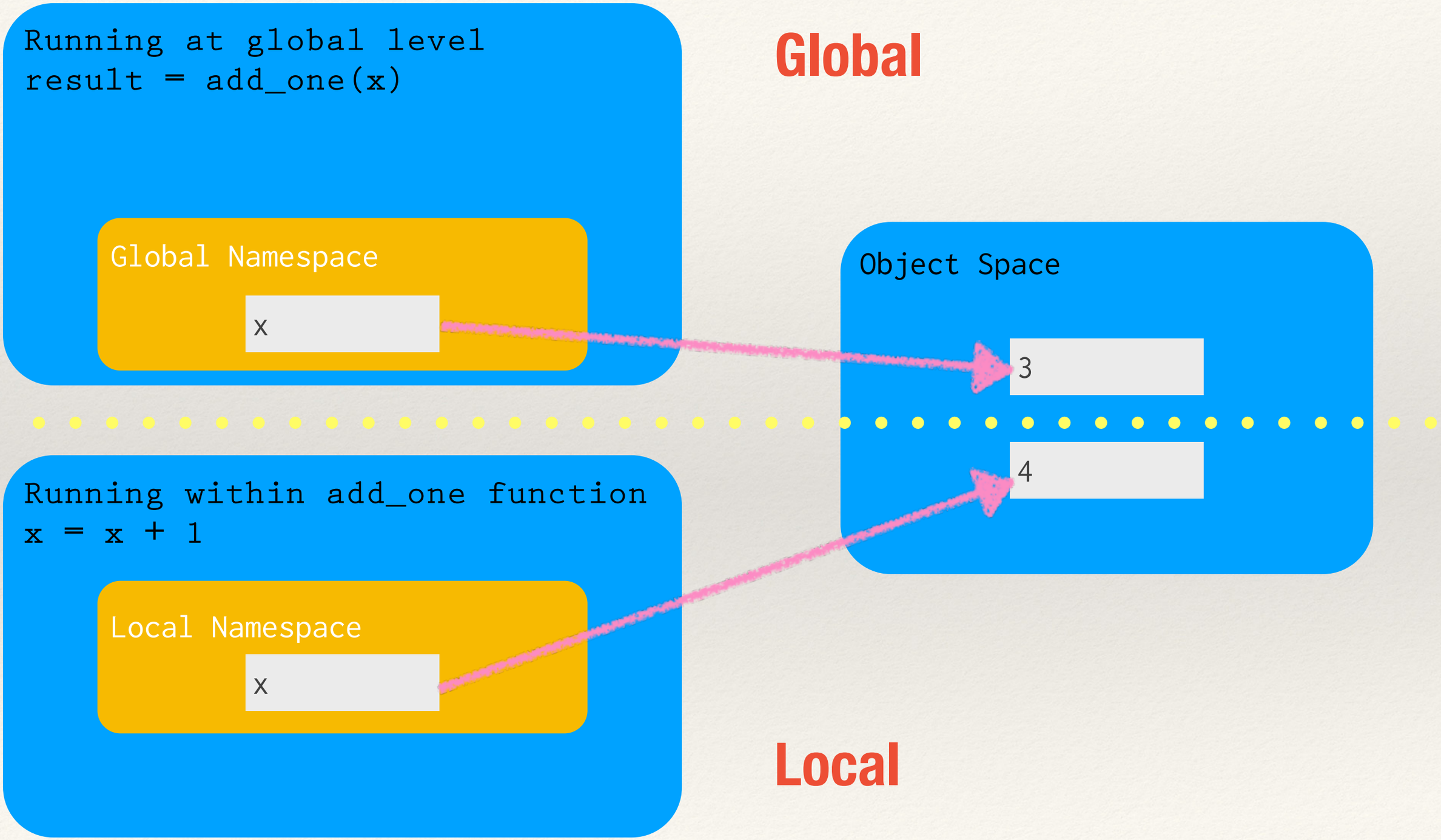
Discuss Homework 4

UC Berkeley
SCHOOL OF INFORMATION

- Remember to use either all tabs or 4 spaces [not both]
- The usual style is to have a space before/after operands
  - (e.g., `a = b + c`, not `a=b+c`)
- Comments are encouraged – usually put them on their own lines, not append to the ends of lines.
- When submitting the final version of homework, delete any personal debugging statements.

Discuss Name Space Concept

UC Berkeley
SCHOOL OF INFORMATION



Discuss Name Space Concept

UC Berkeley
SCHOOL OF INFORMATION

- Discussion: Separate namespaces?
  - What are local variables used for? [examples?]
  - What are global variables used for? [examples?]
- What happens when a variable is referenced but not defined in the local namespace?

Anatomy of a Function

UC Berkeley
SCHOOL OF INFORMATION

- Vocabulary:

1. `my_function(x)` accepts "arguments" also known as "parameters"

2. docstring # help text [readability, reusability]

3. code reuse; modularity; abstraction

4. functions are objects

5. functions are <u>not</u> executed until called by the script

6. Not all functions return a value, but if they do ... look for `return`

Anatomy of a Function

UC Berkeley
SCHOOL OF INFORMATION

```
## square root algorithm as a function

def sqrt(x, epsilon):
    """Newton's method to find square root with precision
       epsilon (Heron's algorithm)"""

    ans = 1
    num_guesses = 0
    while abs(x/ans - ans) > epsilon:
        ans = (x/ans + ans)/2
        num_guesses += 1
    return ans
```

Anatomy of a Function

UC Berkeley
SCHOOL OF INFORMATION

1 define the function

3 arguments become parameters inside function

```
def distance_to_origin(x,y):
    """find the distance from point at (x,y) to the origin"""
    ans = sqrt(x**2 + y**2, 0.00001)
    return ans

magnitude = distance_to_origin(x,y)
```

4 internal operations

5 return variable

2 execute function with parameters

6 assignment to var outside of the function

This may be already clear to you.
Thinking about efficiencies, reflect on the number of steps involved from the computer's p.o.v.

Functions as Objects

UC Berkeley
SCHOOL OF INFORMATION

- Python sees everything as an object (even if our other experience makes us question the syntax)
- And we can query Python to query itself!
  - It has a type

```
>>> type(round10)
>>> function
```

```
a = round10
a(12)
 10
```

  - And we can bind it to a variable name. Can be passed as an argument [perform operation (function) on iterable (example: `grade_list`)]:

```
def apply_to_grades(operation, grade_list):
    return [(name, operation(grade)) for name, grade in grade_list]

grade_list = [(“Betty”, 88), (“Steve”, 75), (“Bob”, 73), (“Ming Wa”, 75)]
print( apply_to_grades(round10, grade_list) )
```

map()

UC Berkeley
SCHOOL OF INFORMATION

- map() executes a specified function for each in an iterable. The item is sent to the function as a parameter:
  - `map(function, iterables)`

Calculate the length of each word in the tuple:

```
def myfunction(n):
    return len(n)

x = map(myfunction, ('Boston', 'Back Bay', 'Cambridge'))
print(list(x))
```

Outputs: [6, 8, 9]

Make new combinations by providing 2 iterable objects:

```
def myfunction(a, b):
    return a + b

x = map(myfunction('a', 'b', 'c'), ('1', '2', '3'))
# convert the map into a list for readability:
print(list(x))
```

Outputs: ['a1', 'b2', 'c3']

lambda

UC Berkeley
SCHOOL OF INFORMATION

- A lambda function is a small, anonymous function, taking any number of arguments but can have only one expression:
  - `lambda` *`arguments`* `:` *`expression`*

Lambda that adds 10 to number passed and prints result:

```
x = lambda a : a + 10
print(x(5))
```

Outputs: 15

Lambda that sums argument a, b, c and prints result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Outputs: 13

Lambda passed as an argument:

```
apply_to_grades(lambda x : 100 - (100 - x)/2, grade_list)
```

Outputs:
```
[('Betty', 94.0),
 ('Steve', 87.,5),
 ('Bob', 86.4),
 ('Ming Wa', 97.0)]
```

map() & lambda

UC Berkeley
SCHOOL OF INFORMATION

- Common to combine both `map()` & `lambda`

```
list(map(lambda x: x**2, (23, 25, 52, 66)))
```

function

iterable

Outputs: `[529, 2025, 1764, 4356]`

Default Arguments

UC Berkeley
SCHOOL OF INFORMATION

- What does it mean for code to be “brittle” or “fragile”?
- Special methods increase flexibility [default types, None values]

```
def feedback(grade=None, comment=None):
    text = “” if comment == None else “ - “ + comment
if grade == None:
    return(“Grade is missing. “ + text)
elif grade >= 90 and grade <= 100:
    return (“A “ + text)
elif grade >= 80 and grade < 90:
    return(“B “ + text)
elif grade >= 70 and grade < 80:
    return(“C “ + text)
elif grade >= 60 and grade < 70:
    return(“D “ + text)
else:
    return(“F “ + text)
```

Outputs:

```
print(feedback(80))
B

print(feedback(75, ‘Please study more’))
C - Please study more

print()
Grade is missing.
```

Default Arguments

UC Berkeley
SCHOOL OF INFORMATION

- Default values won't be reset, if called a second time!
- "Permanent" attributes of a function [in the global namespace]
- Once used, modified.

```
def add_total(order_list = []):
    total = sum([quantity for name, quantity in order_list])
    order_list.append( ("Total", total) )
    print(order_list)
```

Outputs:

```
add_total1()
[('Total', 0)]

add_total1()
[('Total', 0), ('Total', 0)]
```

Positional Arguments

UC Berkeley
SCHOOL OF INFORMATION

- Arguments usually use positional cues …
  - We can define in any order if we specify keywords…

```
print(feedback(90, comment="Keep it up!"))
```

Outputs: `A - Keep it up!`

```
print(feedback(comment="Not bad.", grade=88))
```

Outputs: `B - Not bad.`

System Arguments

UC Berkeley
SCHOOL OF INFORMATION

- So far, just talked about arguments we've passed in Python … but … lots of need for passing arguments from the Operating System (O/S) … the "argument vector", usually `argv`
- Accessible through the library that lets python communicate with the O/S (`import sys`).

System Arguments

UC Berkeley
SCHOOL OF INFORMATION

Demo script in python:

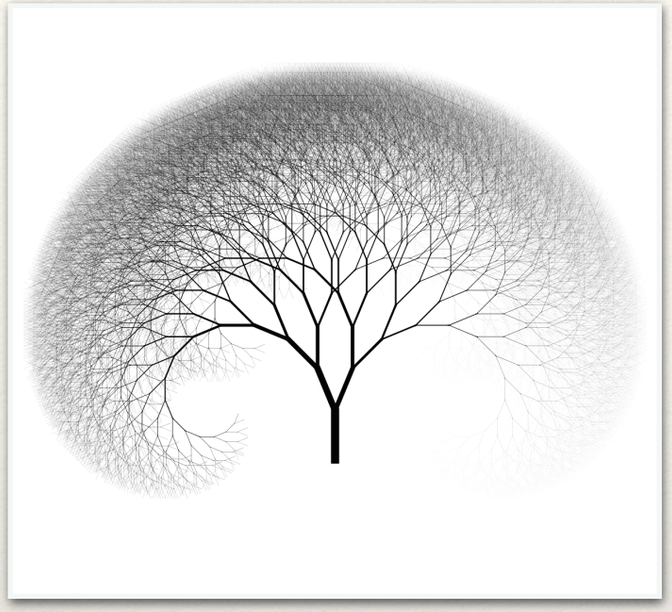
```
import sys

print(sys.argv)

if len(sys.argv) > 1:
    name = sys.argv[1]
else:
    name = input(“Enter your name: “)

for i in range(len(name), 0, -1):
    print( name[0:i], end = “ “)
    for j in range(i, len(name)):
        print(“ “ * (j-1) +
        name[j], end = “”)
    print(“”)
```

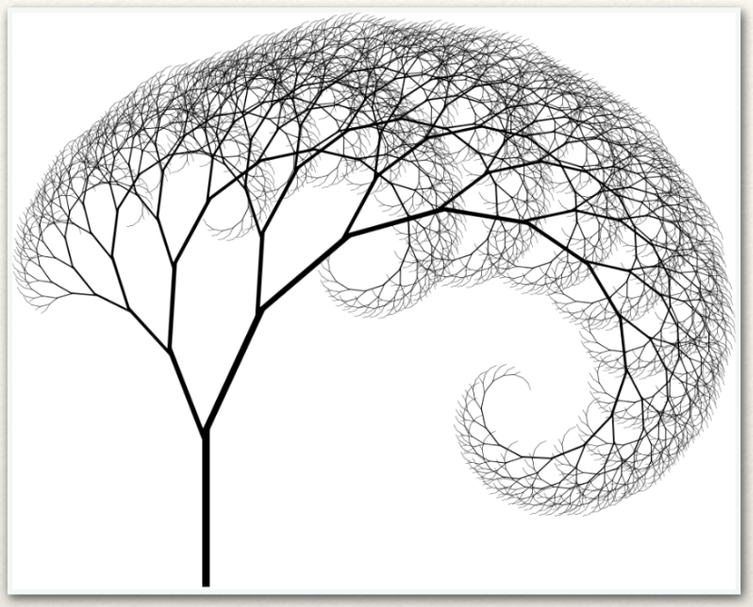
Outputs:

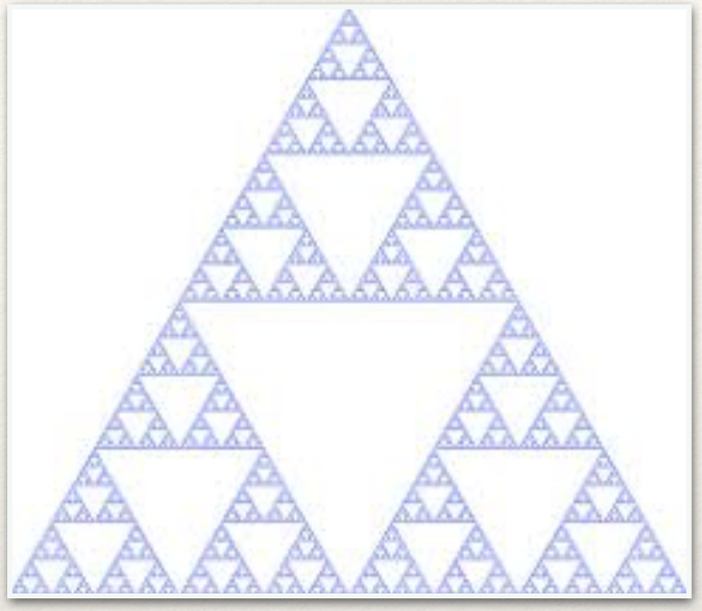
```
>>> import sys
>>> print(sys.argv)
['']
>>> if len(sys.argv) > 1:
...     name = sys.argv[1]
... else:
...     name = input("Enter your name:
")
...
Enter your name: Fifi
>>> for i in range(len(name), 0, -1):
...     print(name[0:i], end = " ")
...     for j in range(i, len(name)):
...         print(" " * (j-1) + name[j],
end = "")
...     print("")
...

Fifi
Fif   i
Fi  f  i
F i f  i
>>>
```

Recursion

UC Berkeley
SCHOOL OF INFORMATION



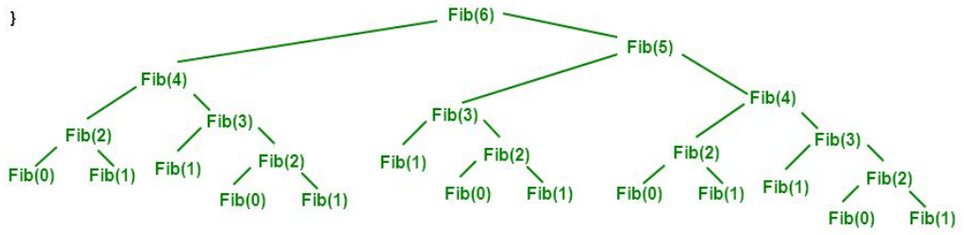




Excessive Recursion: Example

- To show how much this formula is inefficient, let us try to see how Fib(6) is evaluated.

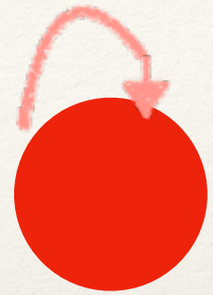
```
int fib(int n) {
  if (n<2)
    return n;
  else
    return fib(n-2)+Fib(n-1);
}
```



Recursion

UC Berkeley
SCHOOL OF INFORMATION

What is recursion?

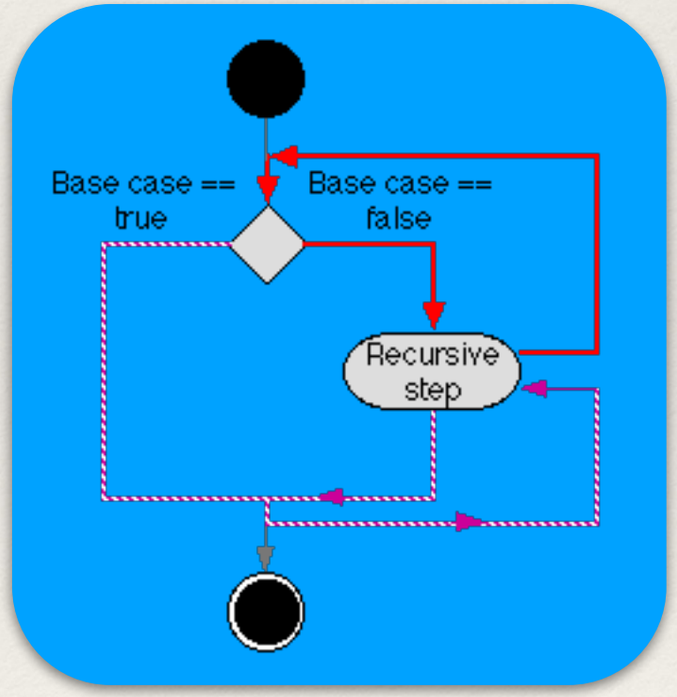


Recursion is a function calling itself.

To use recursion, we must specify:

1) a base case and
2) a recursive rule.

The base case ends the process of the recursive rule (loop) calling itself.



Recursion

UC Berkeley
SCHOOL OF INFORMATION

- Identify the "base case" and the "recursive rule". Why do we need each?

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

Recursion

UC Berkeley
SCHOOL OF INFORMATION

- Identify the "base case" and the "recursive rule". Why do we need each?

Base case →

Recursive rule →

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

One reason to think about recursion is for algorithm efficiency. Calling the same algorithm can increase operating time $O(n) + xn^2$ *where n is the number of recursions and x is the # of steps.*

Recursion

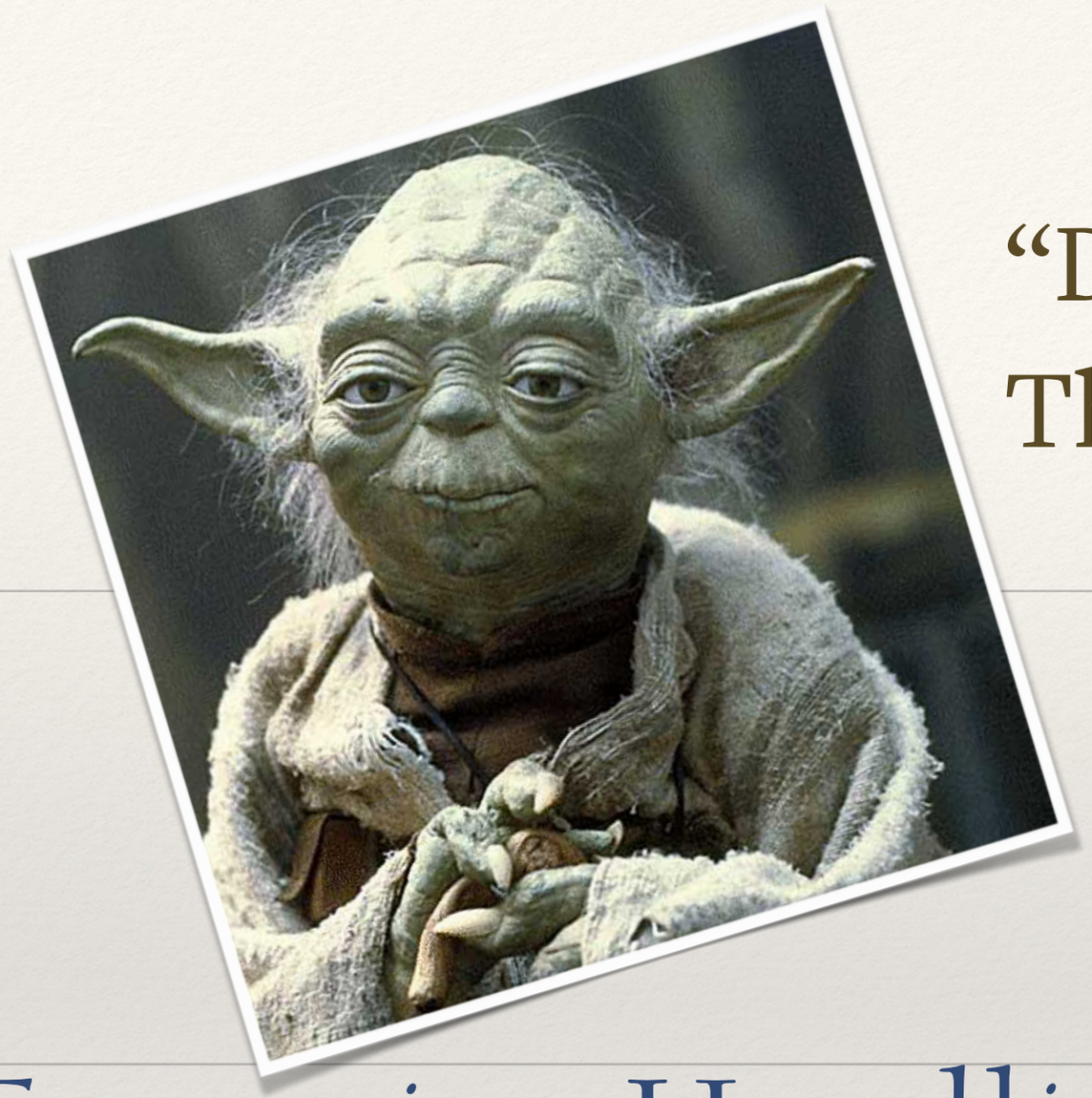
UC Berkeley
SCHOOL OF INFORMATION

`Recursion` vs. `Loop`

Recursion allows us to know "even less" about the structure of a problem. E.g., we can traverse interesting data structures, such as trees and .json, without knowing about them in advance.

We've discussed that a "while" loop allows us to run a single loop an unknown number of times.

Recursion allows us to run an unknown number of loops, an unknown number of times.



“Do or do not!
There is no try!”

Exception Handling

[Is Yoda anticipating try ... except block!?]

Exception Handling . . . *try & except*

UC Berkeley SCHOOL OF INFORMATION

Checking & failing gracefully: Do it all, or don't do any of it, and prepare for any type of error. Our buddies try … except.

```
try:
    x = float(input("Enter a number: "))
    print("The reciprocal is ", 1/x)
except ValueError:
    print("Sorry, your input was not valid.")
except ZeroDivisionError:
    print("Sorry, zero doesn't have a reciprocal")
except:
    Print "Yikes, something else is wrong."
```

specific

general

Notice we have both "general" and "specific" exceptions. Very useful for checking SQL, file access, insertion, user/file input, end-of-file (EOF), file not found (FNF), etc.!

Exception Handling . . . raise

UC Berkeley
SCHOOL OF INFORMATION

- Programming languages that support `try` ... generate an error as a class, an "exception" class. This class has methods to extract the type of error.
- When the exception (e) is raised, we capture it and convert the error (e) to a string [for us humans]...

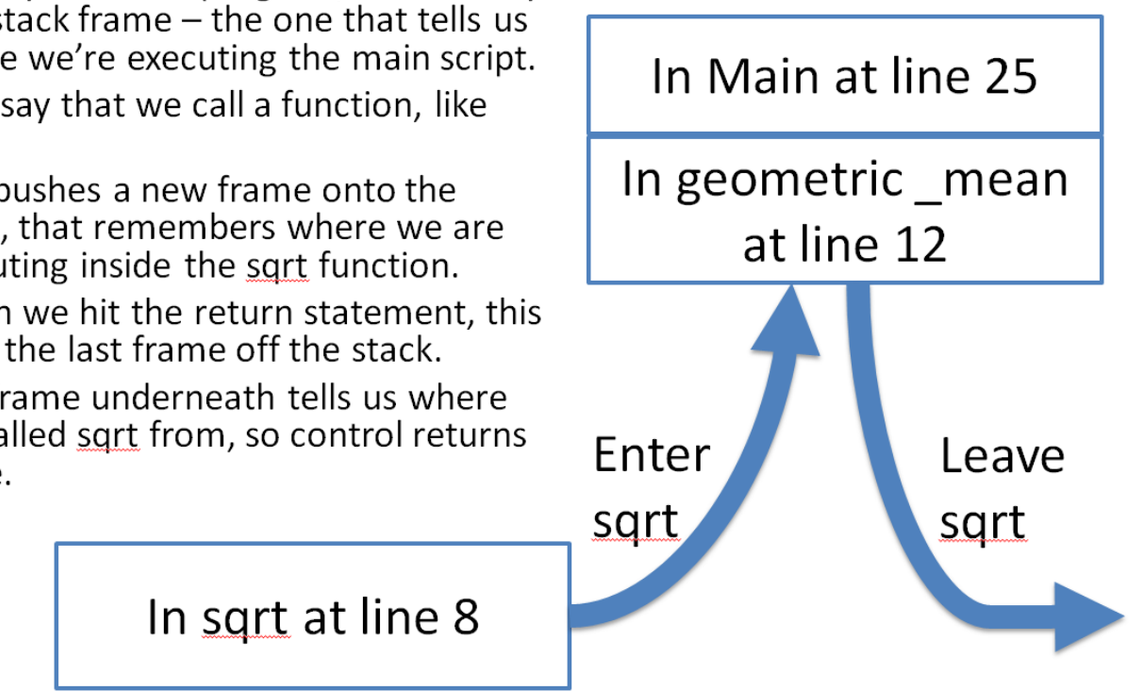
```
try:
    sell("oranges", 15, inventory)
except Exception as e:
    print("Sorry, not enough to sell. " + str(e))


def sell(item, quantity, inventory):

    if item not in inventory:
        raise Exception(str(item) + " doesn't appear in inventory")
    q = inventory[item]

    if q < quantity:
        raise Exception("Sorry, not enough to sell.")
    inventory[item] = q - quantity
```

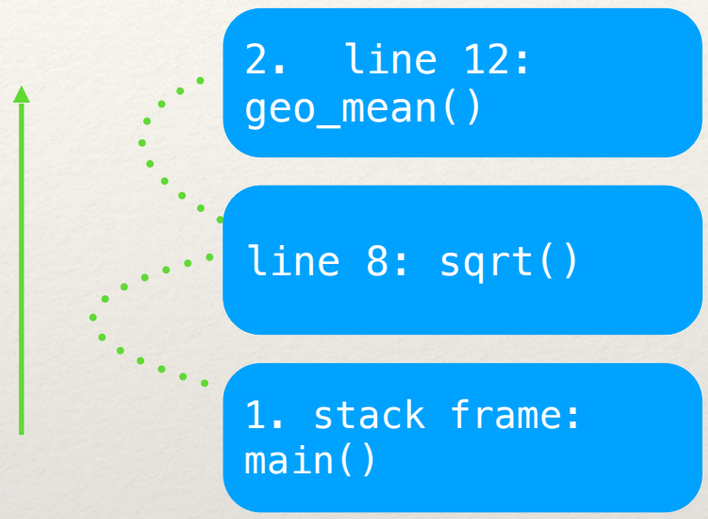
Stack Trace

UC Berkeley
SCHOOL OF INFORMATION

Call Stack

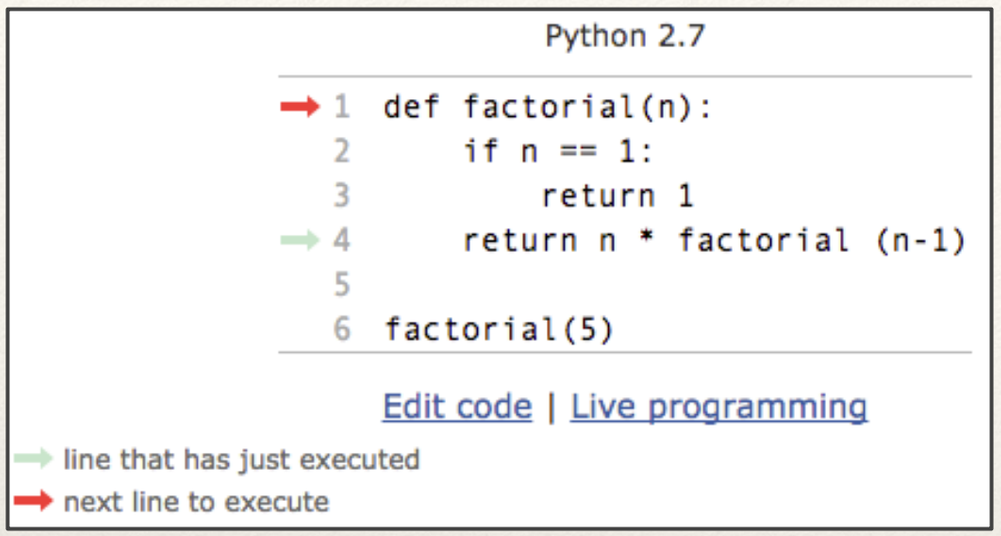
- When you start a program, there's only one stack frame – the one that tells us where we're executing the main script.
- Let's say that we call a function, like sqrt.
- This pushes a new frame onto the stack, that remembers where we are executing inside the sqrt function.
- When we hit the return statement, this pops the last frame off the stack.
- The frame underneath tells us where we called sqrt from, so control returns there.



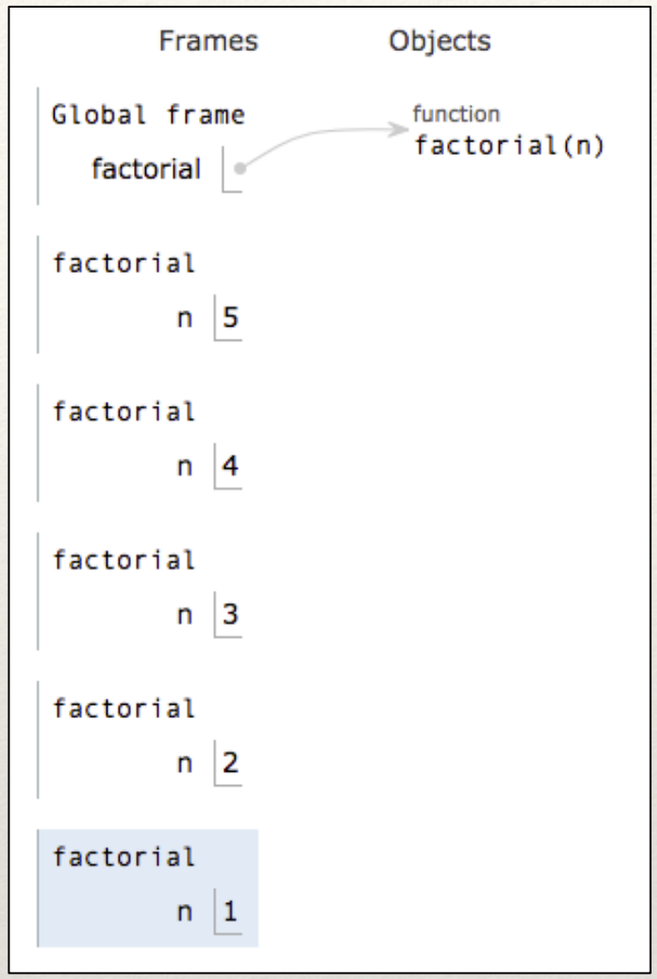


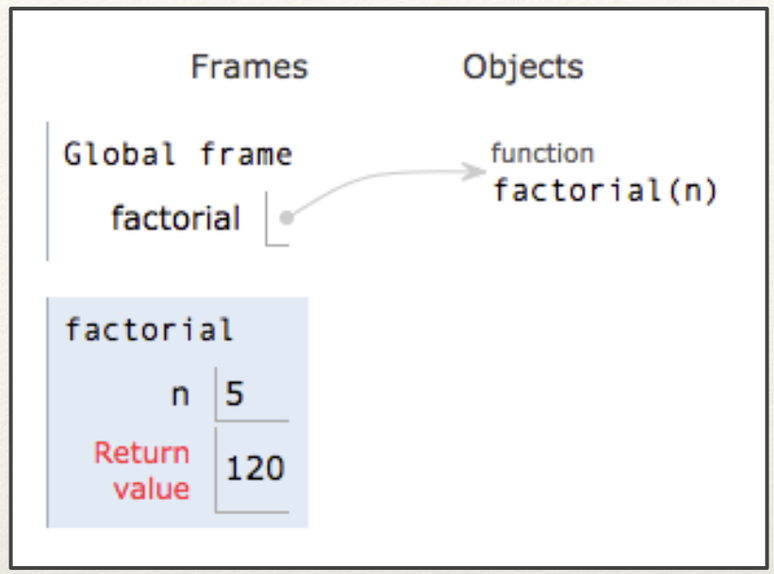
https://docs.python.org/3/library/traceback.html

Stack Trace … recursion



https://goo.gl/bwpHPo





Let's visit http://www.pythontutor.com/visualize.html#mode=display to see the live steps.

Stack Trace . . . errors

UC Berkeley
SCHOOL OF INFORMATION

```
def print_hello(var):
    print("Hello!")
    x = 7 / var
    return x

def some_function(var):
    print("I am the function lord.")
    print(1 + 7 / 3)
    y = print_hello(var)
    print(y)

    return y
```

```
some_function(0)
```

```
I am the function lord.
3.3333333333333335
Hello!

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most r
<ipython-input-11-febbbdbb6e39> in <module>()
----> 1 some_function(0)

<ipython-input-10-3e7fc44e144f> in some_function(var)
      7     print("I am the function lord.")
      8     print(1 + 7 / 3)
----> 9     y = print_hello(var)
     10     print(y)
     11

<ipython-input-10-3e7fc44e144f> in print_hello(var)
      1 def print_hello(var):
      2     print("Hello!")
----> 3     x = 7 / var
      4     return x
      5

ZeroDivisionError: division by zero
```

Lorem Ipsum Dolor

Backup Slides

git hub | branching & merging

UC Berkeley
SCHOOL OF INFORMATION

- Make the branch:

```
git checkout -b <name>      # to add a new branch

git branch                  # tells you what branches there are.
```

- Merge into the branch:

```
git checkout master         # change to branch you want to merge

git merge <alt branch>      # "fast forward" merge indicates no conflicts.
```

Functions | map & lambda

UC Berkeley
SCHOOL OF INFORMATION

- map() `list(map(round10, (23,24,42,66)))` `[20, 50, 40, 70]`
  - Apply a function to each element of an iterable …
- lambda functions (“anonymous”)
  - The functions aren’t named **`lambda x : 100 – (100 – x)/2`**

```
<function __main__.<lambda>>
```

- We can pass the lambda as an argument

```
apply_to_grades(lambda x : 100 – (100 – x)/2, grade_list)
```

```
[(‘Betty’, 94.0),
 (‘Steve’, 87.,5),
 (‘Bob’, 86.4),
 (‘Ming Wa’, 97.0)]
```

btw . . .

UC Berkeley
SCHOOL OF INFORMATION

- if you know other languages, you'll have encountered try ... catch. This is the same thing, with a different syntax.

```
class myClass extends File throws Exception

try {
    fh = read("myfile.txt");
} catch(Exception e) {
    system.out.println("Error: "+e.getMessage())
}
```

Summary

UC Berkeley
SCHOOL OF INFORMATION

- Coming Weeks
- Scope & Visibility of variables
- Functions, with defaults, as objects
- map and lambda
- recursion
- try ... except
- argv and stack trace