Objectives

- Introduction to Recursion
- Comparing Programming Languages

Review: Extensions to Solution

```
def search(searchlist, key):
    low=0
    high = len(searchlist)-1
    while low <= high :
        mid = (low+high)//2
        if searchlist[mid] == key:
            return mid
        elif key > searchlist[mid]:
            # look in upper half
            low = mid+1
        else:
            # look in lower half
            high = mid-1
    return -1
```

Goal: find a Person with a certain name
Consider what happens when **searchlist** is a list of *Persons*, ***key*** is a ***str*** representing the *name*

Good capstone problem:
Brings together

- Algorithms
- Classes/Objects
- Lists
- Methods
- While loops
- Strings

0	1	2	3	4
Person Id: "4" "Ben"	Person Id:"3" "Brie"	Person Id: "1" "Gal"	Person Id:"2" "Henry"	Person Id:"5" "Samuel"

Solving Binary Search

- Our solution was an *iterative* solution
- We could write it as a *recursive* solution
- ***Recursion***: method of solving problems
  - Break a problem down into smaller subproblems of *the same problem* until problem is small enough that it can be solved trivially
- How can we break binary search into smaller problems of the same problem?

Toward Recursive Binary Search

```
def search(searchlist, key):
      mid = len(searchlist)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         return search( searchlist[mid+1:], key )
      else:
         # look in lower half
         return search( searchlist[:mid], key )
```

Toward Recursive Binary Search

```
def search(searchlist, key):
      mid = len(searchlist)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         return search( searchlist[mid+1:], key )
      else:
         # look in lower half
         return search( searchlist[:mid], key )
```

The function calls *itself* on a list that is ~half the size of the original!

Recursion

Recursion: Breaking problem into smaller subproblems of the same problem … into trivially solvable problem

Toward Recursive Binary Search

```
def search(searchlist, key):
      mid = len(searchlist)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         return search( searchlist[mid+1:], key )
      else:
         # look in lower half
         return search( searchlist[:mid], key )
```

The function calls *itself* on a list that is ~half the size of the original!

When does the function stop? It keeps calling itself!
What is the trivial problem we're trying to break down to?

Toward Recursive Binary Search

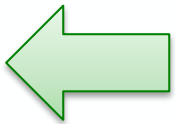
```
def search(searchlist, key):
      mid = len(searchlist)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         return search( searchlist[mid+1:], key )
      else:
         # look in lower half
         return search( searchlist[:mid], key )
```

Stops when we find the element

But, what if the element isn't in the list?
When will we know that?

Recursive Binary Search

```
def search(searchlist, key):
    if len(searchlist) == 0:
        return -1
    mid = len(searchlist)//2
    if searchlist[mid] == key:
        return mid
    elif key > searchlist[mid]:
        # look in upper half
        return search( searchlist[mid+1:], key )
    else:
        # look in lower half
        return search( searchlist[:mid], key )
```



Base case: We know the key is not in our list

Recursive Binary Search Conclusions

```
def search(searchlist, key):
    if len(searchlist) == 0:
        return -1
    mid = len(searchlist)//2
    if searchlist[mid] == key:
        return mid
    elif key > searchlist[mid]:
        # look in upper half
        return search( searchlis
    else:
        # look in lower half
        return search( searchlist[:mid], key )
```

- Broke problem into smaller problems of *same* problem
- Smallest problem is easy to solve

- BUT, this is not an efficient solution because creates multiple lists
  - We can write a recursive solution that doesn't create multiple lists but would need to change the function signature.

More Efficient Recursive Binary Search

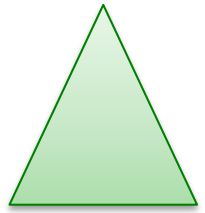
```
def search(searchlist, key, low, high):
    if high < low:
        return -1
    mid = len(searchlist)//2
    if searchlist[mid] == key:
        return mid
    elif key > searchlist[mid]:
        # look in upper half
        return search( searchlist, key, mid+1, high )
    else:
        # look in lower half
        return search( searchlist, key, low, mid-1 )
```

Does not create additional lists
Just narrows range of values under consideration

Initial call: `search( searchlist, key, 0, len(searchlist)-1)`

Recursion Summary

- ***Recursion***: method of solving problems
  - Break a problem down into smaller subproblems until problem is small enough that it can be solved trivially
- Binary Search:
  - Break problem to ~half the size of original problem
  - Base cases: when the middle element is what you're looking for; when there are no elements in your list
- Any recursive problem can be solved iteratively
  - Some problems lend themselves better to recursive solutions



COMPARING PROGRAMMING LANGUAGES

Applying What You Know To Other Languages

- At the beginning of the semester, some of you wondered
  - "Why the Python programming language?"
  - "Will I be able to read/write programs in other programming languages?"
- We'll answer the first question by showing that you can do the second

Review:
Programming Language Characteristics

- **Syntax**: symbols used
- **Semantics**: what the symbols *mean*

What is the Python 3 Program Doing?

What is the Python3 Program Doing?

- Getting a line of input from "standard in" (from the user)
- Splitting the input into integers
- Calculating the result of a formula
- Deciding if a student is admitted, based on the result of the formula
- Displaying the result

Admissions Problem

- Binary University decides to admit students based on a formula that weighs various factors
  - Scores of 70 or better are admitted
- Input: single line, 4 integers, in order below

Category	Range	Weight Factor (Multiplier)
AP Courses	0-10	3
Intangibles	1-10	2
High School GPA	0 - 100	0.25
SAT score	400-1600	.02

Example Input/Expected Output

Input	Expected Output
0 1 0 300	DENY
6 10 99 1590	ADMIT
0 7 82 1500	ADMIT
2 5 80 990	DENY
5 5 92 1200	ADMIT
2 5 100 1300	ADMIT

What is the Python3 Program Doing?

- Getting a line of input from "standard in" (from the user)
- Splitting the input into integers
- Calculating the result of a formula
- Deciding if a student is admitted, based on the result of the formula
- Displaying the result

Identify these pieces in the other programs

Comparing Programming Languages

- How is the syntax/semantics of these languages different from Python?

- What is easier or harder to do in these other programming languages than in Python?

Comparing Programming Languages

Benefits of Python

- Simpler syntax (e.g., fewer {} and ())
- Can cover some content with less overhead

Drawbacks

- Data types aren't explicit (static)
  - Can be harder for you to remember and keep straight
- Not compiled explicitly beforehand
  - Keep executing to find all the syntax bugs
  - Doesn't check: "you're passing a file instead of a string"
- Allows you to do some things that won't work in other programming languages

Bash

- Scripting language
  - Can call Unix commands
- Example program:
  - `createPrintableLab`

Tiobe Index

based on the number of skilled engineers world-wide, courses and third party vendors

Apr 2024	Apr 2023	Change	Programming Language	Ratings	Change
1	1	CSCI111, 112	Python	16.41%	+1.90%
2	2	CSCI210, 320	C	10.21%	-4.20%
3	4	^	C++	9.76%	-3.20%
4	3	CSCI209, 335	Java	8.94%	-4.29%
5	5		C#	6.77%	-1.44%
6	7	CSCI335	JavaScript	2.89%	+0.79%
7	10	^	Go	1.85%	+0.57%
8	6	v	Visual Basic	1.70%	-2.70%
9	8	CSCI335, 317	SQL	1.61%	-0.06%
10	20	^^	Fortran	1.47%	+0.88%

https://www.tiobe.com/tiobe-index/

Final Exam

- Final will be in Canvas
  - Take anytime during finals (Saturday 2 p.m. – Friday at noon)
  - Due end of exam period - Friday at noon
- Prep document on schedule
  - Similar format to previous exams but in Canvas
  - More on Friday

Course Evaluations

- On Canvas, due Monday at 11:59 p.m.
- Incentive
  - If 60% of students complete evaluation, 1% Extra Credit on *lab* grades
  - For each additional 10% of students who complete evaluation, additional 1% EC on lab grades
  - Total possible EC: 5%

Extra Credit Opportunity

Evaluate up to 2 games on Canvas for up to 10 points Extra Credit towards labs

Looking Ahead

- Thursday: BI write up due
- Friday:
  - Lab 11 due
  - Review computer science
    - Where we've been and where you can go
  - Bring your exam questions
    - Practice
- All (late) lab work must be submitted by **MONDAY 11:59 p.m.**
- All extra credit articles must be submitted by **FRIDAY 11:59 p.m.**
  - Recommendation: Spend time studying for final exam (worth more)