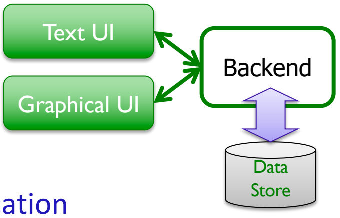
Objectives

- Exception Handling
- Searching

Reviewing Lab 10



Text UI

Graphical UI

Backend

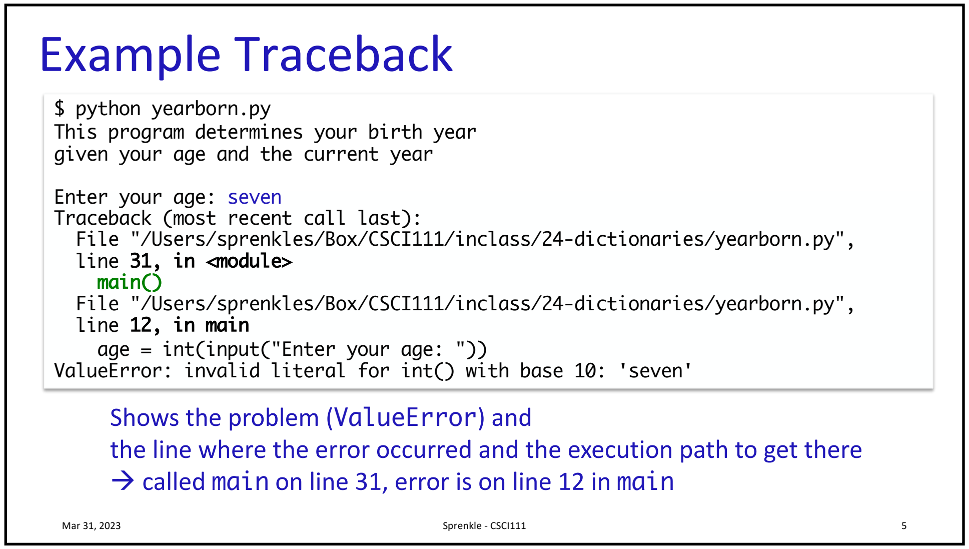
Data Store

- Created two classes
  - Used one class within another class
  - Tested them
  - Example of a backend to a **real** application
    - Could add a different user interface
- “Good judgment comes from experience”
  - Test methods after writing method
  - Remember your data types
  - Refer to the data type’s API
- What could you do to improve your development process?

EXCEPTION HANDLING

Runtime Errors: Exceptions

- “Raised” at runtime
- A signal that something “ain’t quite right”
  - Something has occurred that can’t be easily handled using typical Python structures
- When an exception is raised
  - Program execution stops
  - Python prints out the ***traceback***
    - A report of the function calls made in your code to reach this point

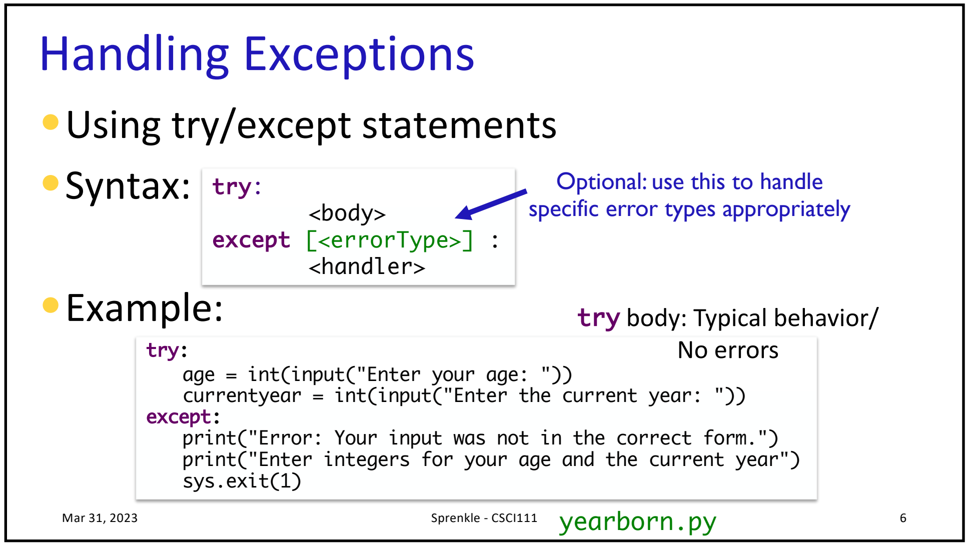


Example Traceback

```
$ python yearborn.py
This program determines your birth year
given your age and the current year

Enter your age: seven
Traceback (most recent call last):
  File "/Users/sprenkles/Box/CSCI111/inclass/24-dictionaries/yearborn.py",
  line 31, in <module>
    main()
  File "/Users/sprenkles/Box/CSCI111/inclass/24-dictionaries/yearborn.py",
  line 12, in main
    age = int(input("Enter your age: "))
ValueError: invalid literal for int() with base 10: 'seven'
```

Shows the problem (`ValueError`) and
the line where the error occurred and the execution path to get there
→ called `main` on line 31, error is on line 12 in `main`



Handling Exceptions

- Using try/except statements
- Syntax:

```
try:
        <body>
except [<errorType>] :
        <handler>
```

Optional: use this to handle specific error types appropriately

- Example:

try body: Typical behavior/ No errors

```
try:
   age = int(input("Enter your age: "))
   currentyear = int(input("Enter the current year: "))
except:
   print("Error: Your input was not in the correct form.")
   print("Enter integers for your age and the current year")
   sys.exit(1)
```

yearborn.py

Exception-Specific Handling

- Using try/except statements
- Syntax:

```
try:
        <body>
except [<errorType>] :
        <handler>
```

Optional: use this to handle specific error types appropriately

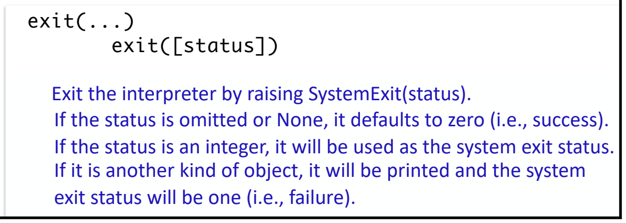
- Example:

```
try:
   age = int(input("Enter your age: "))
   currentyear = int(input("Enter the current year: "))
except ValueError:
   print("Error: Your input was not in the correct form.")
   print("Enter integers for your age and the current year")
   sys.exit(1)
```

yearborn2.py

Discussion: sys.exit([status])

- What is sys.exit([status])?
  - A way to exit the program
- Where does it come from?
  - The sys module; need to import
  - import sys



```
exit(...)
       exit([status])
```

Exit the interpreter by raising SystemExit(status).
If the status is omitted or None, it defaults to zero (i.e., success).
If the status is an integer, it will be used as the system exit status.
If it is another kind of object, it will be printed and the system exit status will be one (i.e., failure).

Examples of Types of Exceptions

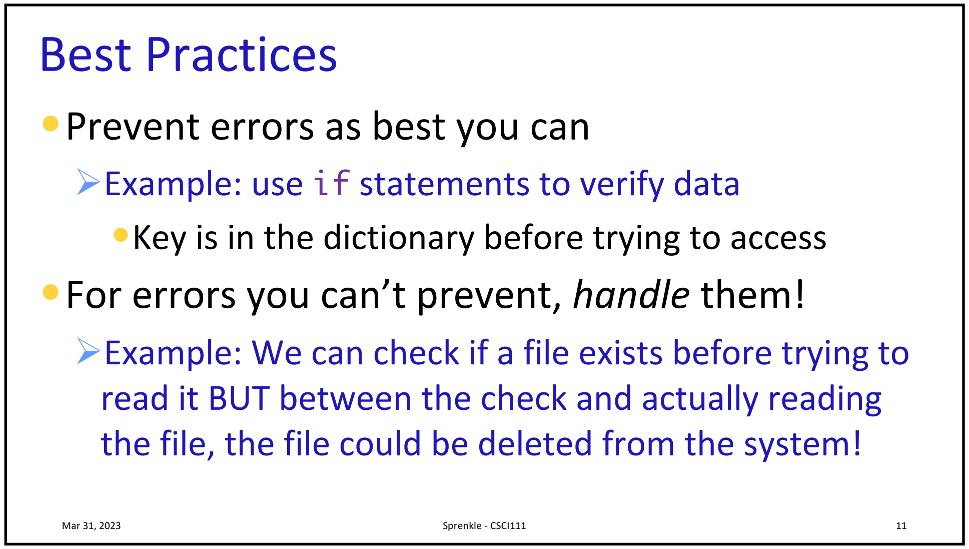
- IndexError
  - When index is not found in the sequence
- KeyError
  - When a key is not found in the dictionary
- IOError:
  - FileNotFoundError: File doesn’t exist
  - PermissionError: Don’t have permission to read/write file

Exception Handling

```
try:
    inFile = open(infileName, "r")
    # normally, would process file here…
    inFile.close()
except IOError as exc :
    print("Error reading \"" + infileName + "\".")
    # could be a variety of different problems,
    # so print out the exception and its type
    print(exc)
    print(type(exc))
    sys.exit(1)
```

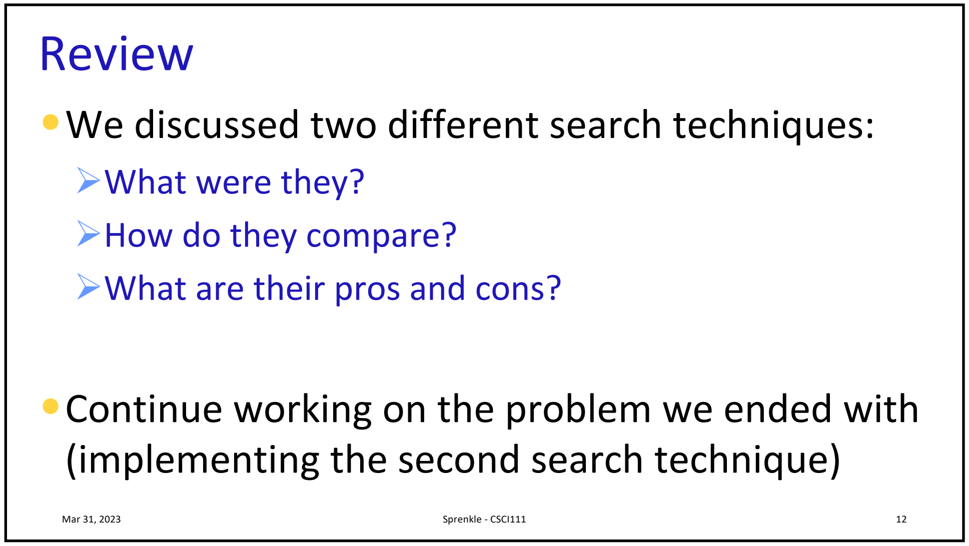
- Exceptions are *objects*
- We can get more information about the exception by printing them out

file_handle.py



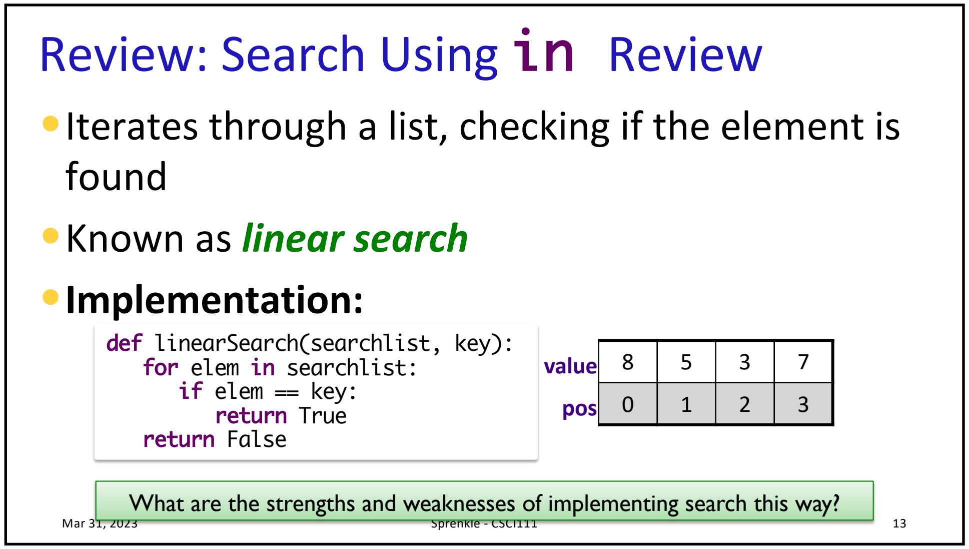
Best Practices

- Prevent errors as best you can
  - Example: use `if` statements to verify data
    - Key is in the dictionary before trying to access
- For errors you can't prevent, *handle* them!
  - Example: We can check if a file exists before trying to read it BUT between the check and actually reading the file, the file could be deleted from the system!



Review

- We discussed two different search techniques:
  - What were they?
  - How do they compare?
  - What are their pros and cons?
- Continue working on the problem we ended with (implementing the second search technique)



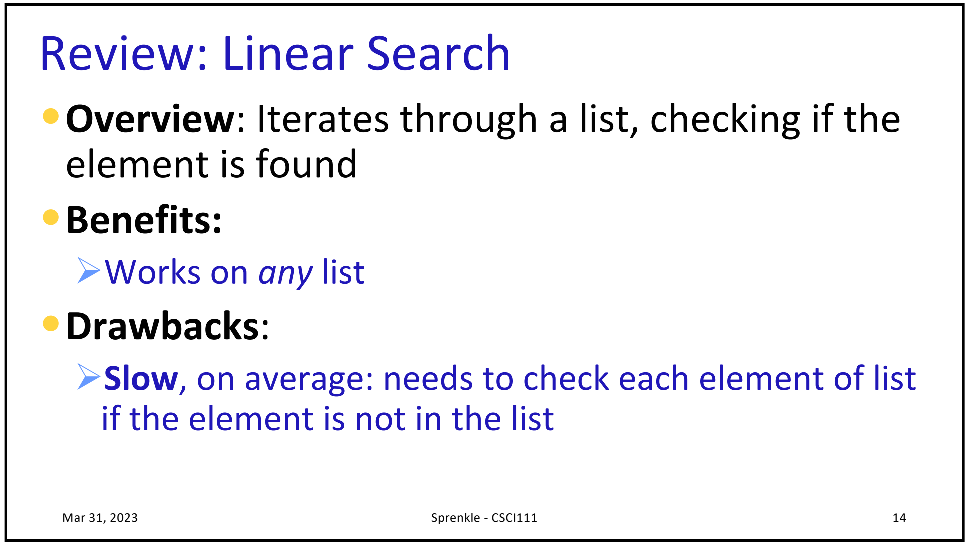
Review: Search Using `in` Review

- Iterates through a list, checking if the element is found
- Known as ***linear search***
- **Implementation:**

```
def linearSearch(searchlist, key):
   for elem in searchlist:
      if elem == key:
         return True
   return False
```

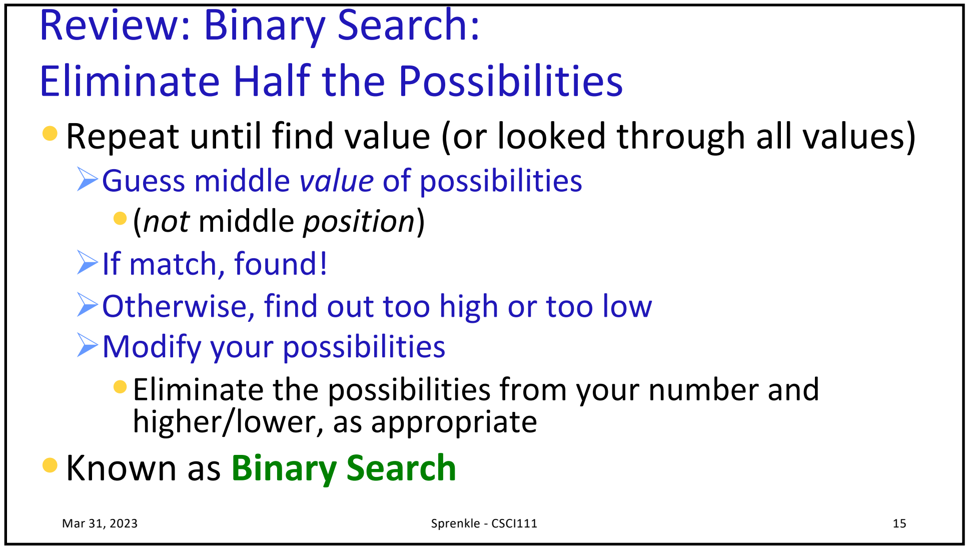
value	8	5	3	7
pos	0	1	2	3

What are the strengths and weaknesses of implementing search this way?



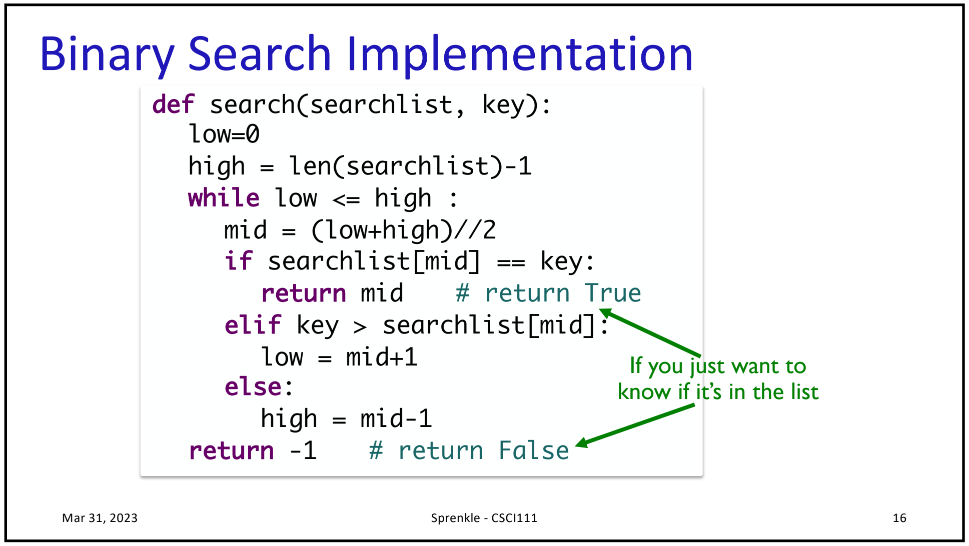
Review: Linear Search

- **Overview**: Iterates through a list, checking if the element is found
- **Benefits:**
  - ➢Works on *any* list
- **Drawbacks**:
  - ➢**Slow**, on average: needs to check each element of list if the element is not in the list



Review: Binary Search: Eliminate Half the Possibilities

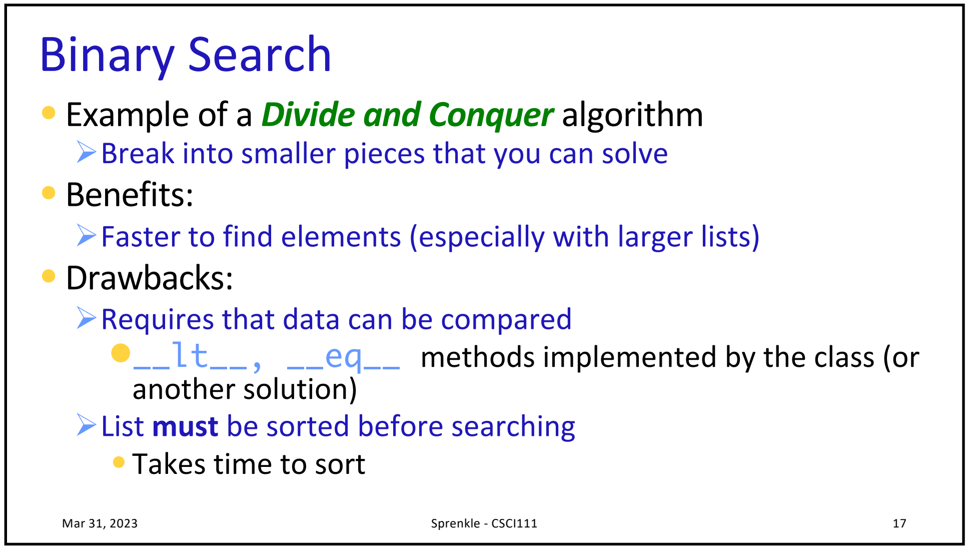
- Repeat until find value (or looked through all values)
  - Guess middle *value* of possibilities
    - (*not* middle *position*)
  - If match, found!
  - Otherwise, find out too high or too low
  - Modify your possibilities
    - Eliminate the possibilities from your number and higher/lower, as appropriate
- Known as **Binary Search**



Binary Search Implementation

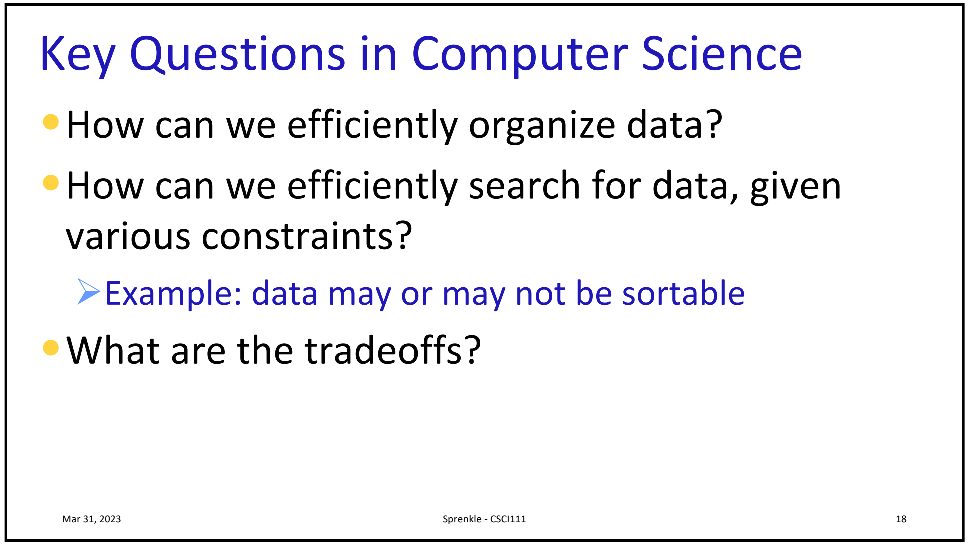
```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while low <= high :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid    # return True
      elif key > searchlist[mid]:
         low = mid+1
      else:
         high = mid-1
   return -1    # return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

If you just want to know if it's in the list



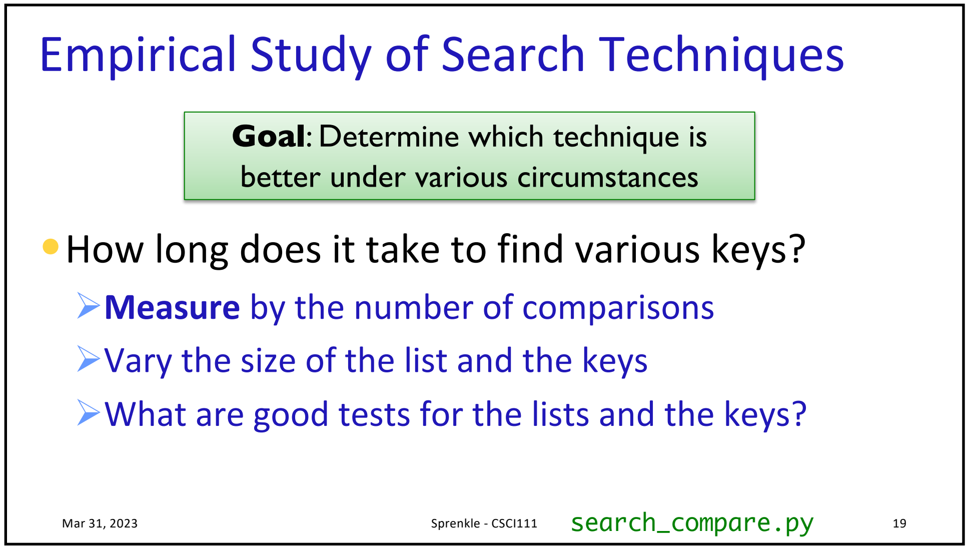
Binary Search

- Example of a ***Divide and Conquer*** algorithm
  - Break into smaller pieces that you can solve
- Benefits:
  - Faster to find elements (especially with larger lists)
- Drawbacks:
  - Requires that data can be compared
    - `__lt__, __eq__` methods implemented by the class (or another solution)
  - List **must** be sorted before searching
    - Takes time to sort



Key Questions in Computer Science

- How can we efficiently organize data?
- How can we efficiently search for data, given various constraints?
  - Example: data may or may not be sortable
- What are the tradeoffs?



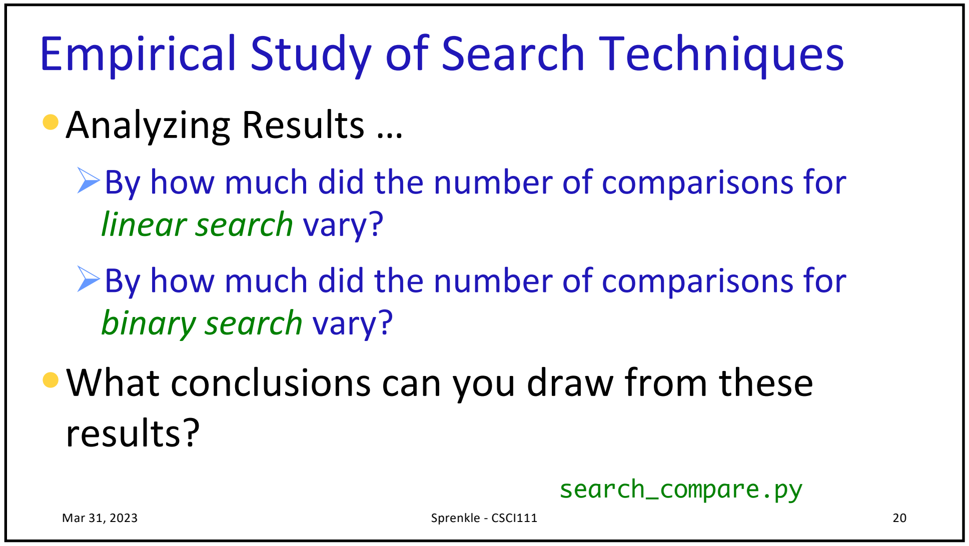
Empirical Study of Search Techniques

Goal: Determine which technique is better under various circumstances

- How long does it take to find various keys?
  - **Measure** by the number of comparisons
  - Vary the size of the list and the keys
  - What are good tests for the lists and the keys?

search_compare.py

19

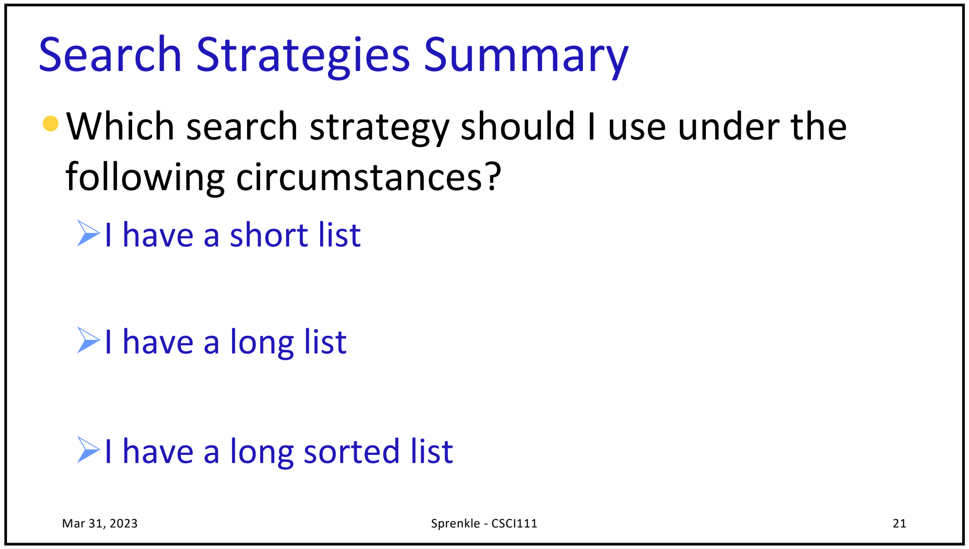


Empirical Study of Search Techniques

- Analyzing Results …
  - By how much did the number of comparisons for *linear search* vary?
  - By how much did the number of comparisons for *binary search* vary?
- What conclusions can you draw from these results?

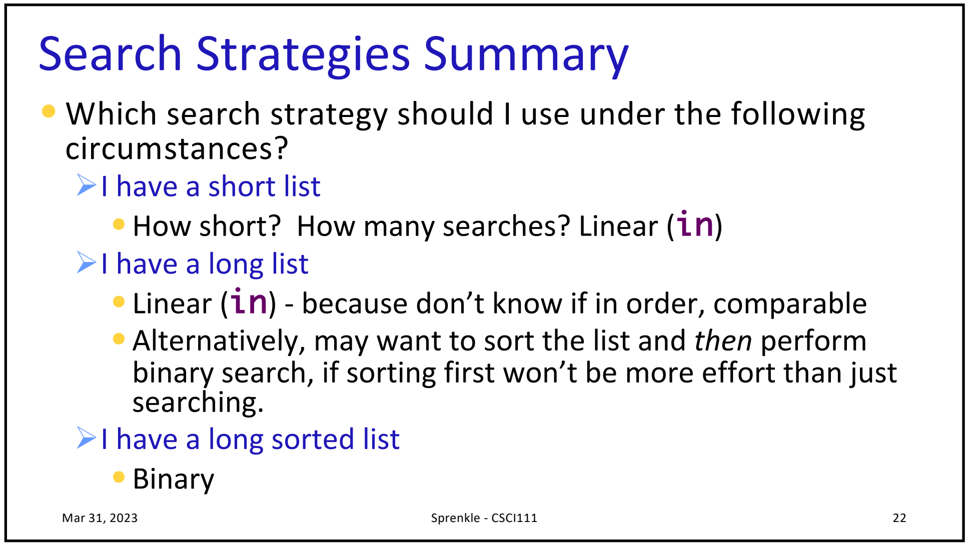
search_compare.py

20



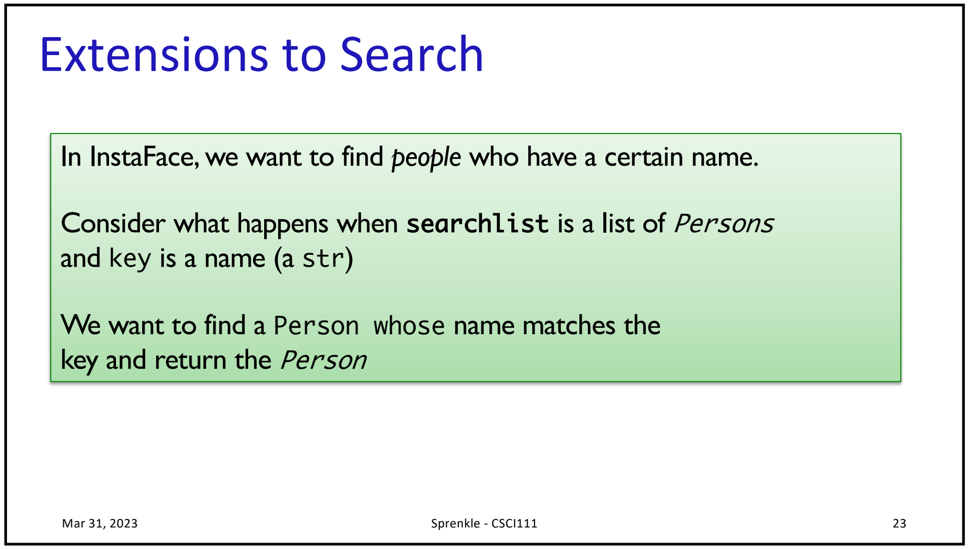
Search Strategies Summary

- Which search strategy should I use under the following circumstances?
  - I have a short list
  - I have a long list
  - I have a long sorted list



Search Strategies Summary

- Which search strategy should I use under the following circumstances?
  - I have a short list
    - How short? How many searches? Linear (`in`)
  - I have a long list
    - Linear (`in`) - because don’t know if in order, comparable
    - Alternatively, may want to sort the list and *then* perform binary search, if sorting first won’t be more effort than just searching.
  - I have a long sorted list
    - Binary

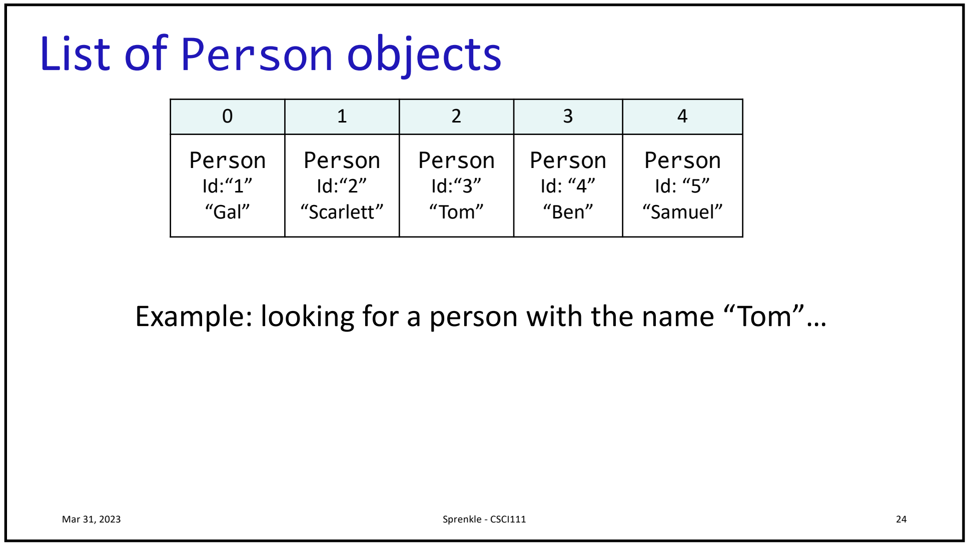


Extensions to Search

In InstaFace, we want to find *people* who have a certain name.

Consider what happens when **`searchlist`** is a list of *Persons* and `key` is a name (a `str`)

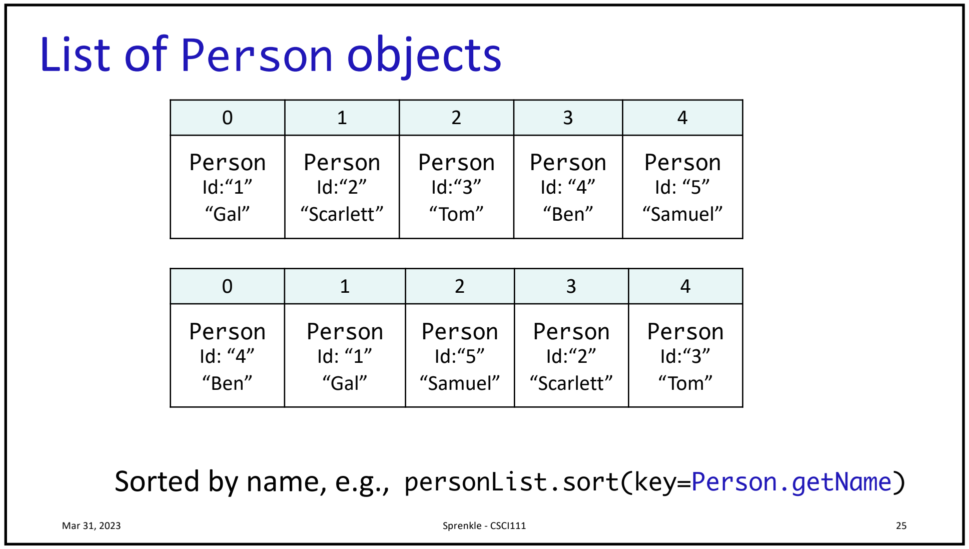
We want to find a `Person` `whose` name matches the key and return the *Person*



List of Person objects

0	1	2	3	4
Person Id:"1" "Gal"	Person Id:"2" "Scarlett"	Person Id:"3" "Tom"	Person Id: "4" "Ben"	Person Id: "5" "Samuel"

Example: looking for a person with the name "Tom"…

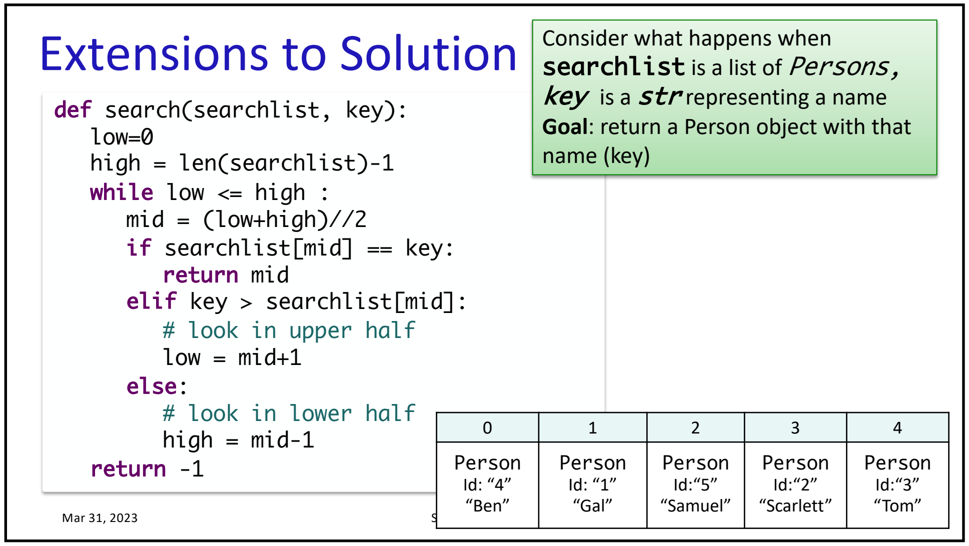


List of Person objects

0	1	2	3	4
Person Id:“1” “Gal”	Person Id:“2” “Scarlett”	Person Id:“3” “Tom”	Person Id: “4” “Ben”	Person Id: “5” “Samuel”

0	1	2	3	4
Person Id: “4” “Ben”	Person Id: “1” “Gal”	Person Id:“5” “Samuel”	Person Id:“2” “Scarlett”	Person Id:“3” “Tom”

Sorted by name, e.g., `personList.sort(key=Person.getName)`

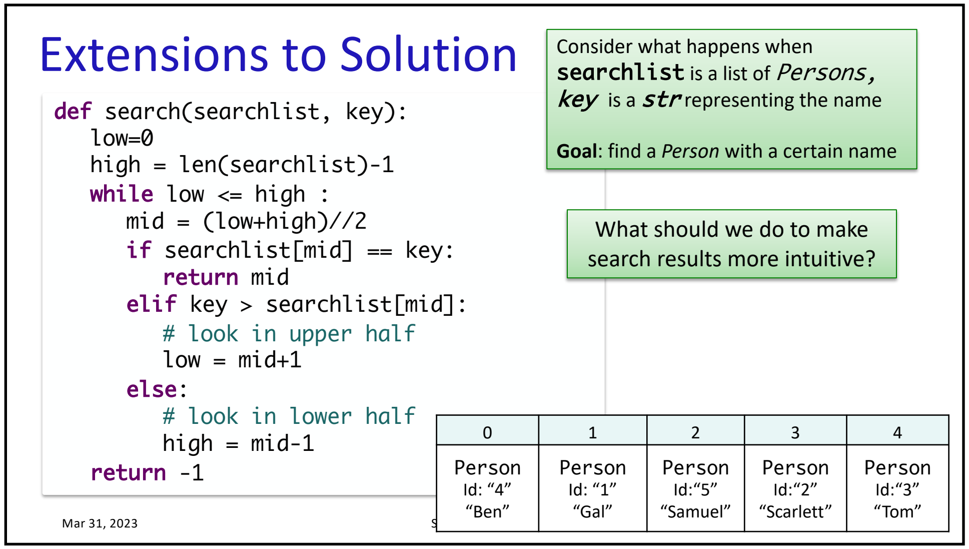


Extensions to Solution

Consider what happens when **`searchlist`** is a list of *`Persons`*, ***`key`*** is a ***`str`*** representing a name
Goal: return a Person object with that name (key)

```
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
```

0	1	2	3	4
Person Id: “4” “Ben”	Person Id: “1” “Gal”	Person Id:“5” “Samuel”	Person Id:“2” “Scarlett”	Person Id:“3” “Tom”



Extensions to Solution

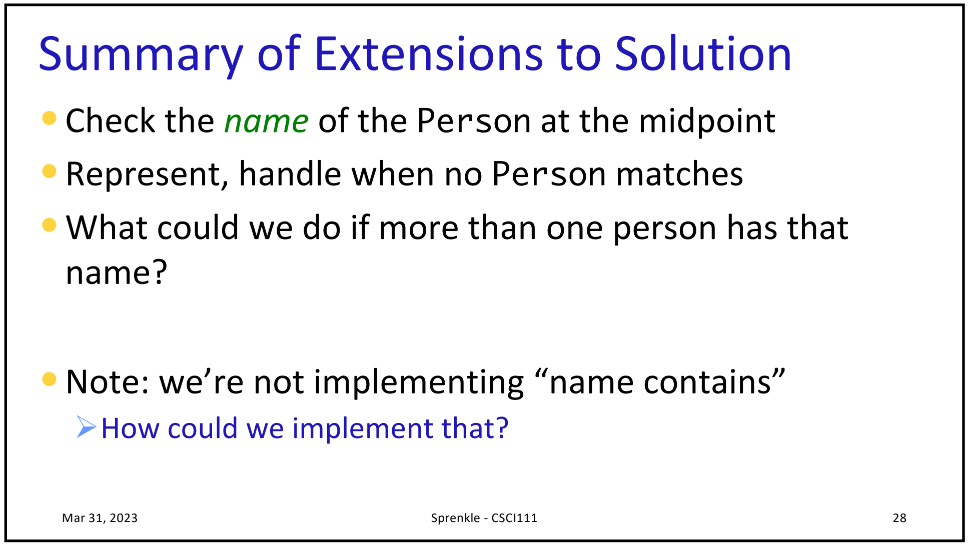
```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while low <= high :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         low = mid+1
      else:
         # look in lower half
 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

Consider what happens when **searchlist** is a list of *Persons*, ***key*** is a ***str*** representing the name

Goal: find a *Person* with a certain name

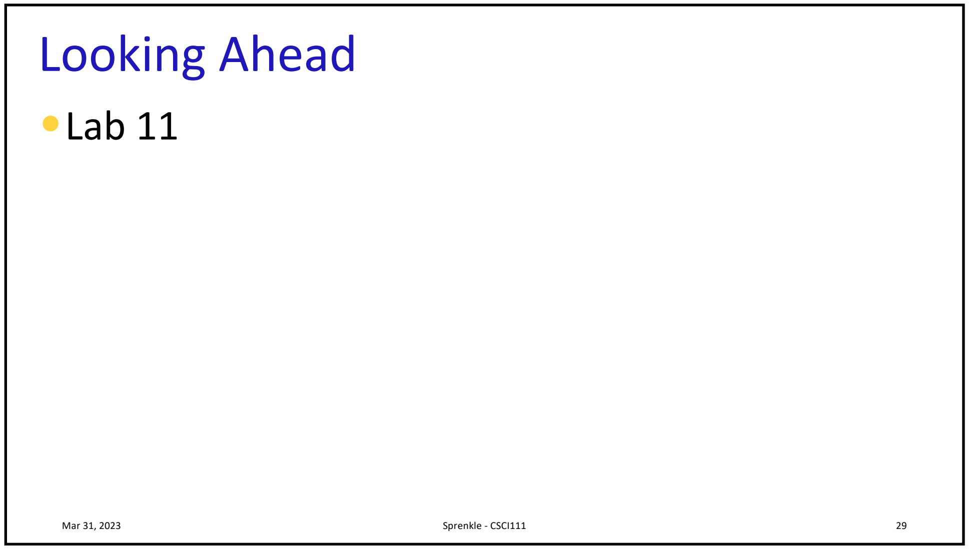
What should we do to make search results more intuitive?

0	1	2	3	4
Person Id: “4” “Ben”	Person Id: “1” “Gal”	Person Id:“5” “Samuel”	Person Id:“2” “Scarlett”	Person Id:“3” “Tom”



Summary of Extensions to Solution

- Check the *name* of the Person at the midpoint
- Represent, handle when no Person matches
- What could we do if more than one person has that name?
- Note: we’re not implementing “name contains”
  - How could we implement that?



Looking Ahead

- Lab 11