Last Class We Covered

• Tuples
• Dictionaries
  – Creating
  – Accessing
  – Manipulating
• Dictionaries vs Lists
Any Questions from Last Time?
Review: Tuples

• Create five tuples about you
  – (e.g., your major is CMSC, your age is 19)

• Create a tuple with all of the courses you’re taking this semester

• Create a tuple with a single element

• Create an empty tuple

• Create a tuple by casting a list
Review: Dictionaries

• Create a dictionary that contains four different (key, value) pairs, similar to “a is for apple”
  – Add one additional (key, value) pair
  – Update one of your (key, value) pairs
  – Remove one of your (key, value) pairs

• Why must dictionary keys be unique?
• Do values need to be unique?
Review: Matching Symbols

• Match the following data types to the symbols needed to create them (may be more than one)

- Dictionary
- List
- String
- Tuple

{ }
( )
" "
[ ]
' '
Review: Matching Symbols

- Match the following data types to the symbols needed to create them (may be more than one)

- Dictionary
- List
- String
- Tuple
Review: Mutability

- Which of the following are mutable data types?

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>???</td>
</tr>
<tr>
<td>Dictionary</td>
<td>???</td>
</tr>
<tr>
<td>Float</td>
<td>???</td>
</tr>
<tr>
<td>Integer</td>
<td>???</td>
</tr>
<tr>
<td>List</td>
<td>???</td>
</tr>
<tr>
<td>String</td>
<td>???</td>
</tr>
<tr>
<td>Tuple</td>
<td>???</td>
</tr>
</tbody>
</table>
# Review: Mutability

- Which of the following are mutable data types?

<table>
<thead>
<tr>
<th>Type</th>
<th>Mutability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Immutable</td>
</tr>
<tr>
<td>Dictionary</td>
<td><strong>Mutable</strong></td>
</tr>
<tr>
<td>Float</td>
<td>Immutable</td>
</tr>
<tr>
<td>Integer</td>
<td>Immutable</td>
</tr>
<tr>
<td>List</td>
<td><strong>Mutable</strong></td>
</tr>
<tr>
<td>String</td>
<td>Immutable</td>
</tr>
<tr>
<td>Tuple</td>
<td>Immutable</td>
</tr>
</tbody>
</table>
Review: Implementation

• You are given a dictionary of the NATO phonetic alphabet, in the form:

```python
alpha = {"A" : "Alpha", "B" : "Bravo", "C" : "Charlie", ... etc.}
```

• Write a function to convert a string from the user into its phonetic code words
  – You need only handle letters (upper and lowercase)
Review: Implementation Example

• Here is an example of how it should work:

Please enter a word: EXAMPLE
The word "EXAMPLE" becomes "Echo X-ray Alpha Mike Papa Lima Echo"

Please enter a word: dogmeat
The word "dogmeat" becomes "Delta Oscar Golf Mike Echo Alpha Tango"
Any Questions about the Material we Just Reviewed?
Today’s Objectives

• To learn more about searching algorithms
  – Linear search
  – Binary search
• To understand why certain algorithms are “better” than others
• To learn about asymptotic performance
  – To examine how fast an algorithm “runs”
Searching

• Sometimes, we use the location of a piece of information in a list to store information

• If I have the list \([4, 5, 2, 3]\), there may be some significance to this order
  – That means sometimes we want to find where in the list something is!
Exercise: Search

• Write a function that takes a list and a variable and returns the first location of the variable in the list
  — If it’s not found, return -1

```python
def find(myList, myVar):
```
def find(myList, myVar):
    for i in range(0, len(myList)):
        if myList[i] == myVar:
            return i
    # we didn't find the variable
    return -1
Linear Search

• This is called linear search!
• It’s a pretty common, simple operation

• It’s especially useful when our information isn’t in a sorted order
Searching Sorted Information

• Now, imagine we’re looking for information in something sorted, like a phone book
• We know someone’s name, and want to find their entry in the book (just like we knew the variable we were trying to locate earlier)

• What is a good algorithm for locating their phone number? Think about how you would do this.
Algorithm in English

• Open the book midway through.
  – If the person’s name is **on** the page you opened to
    • You’re done!
  – If the person’s name is **after** the page you opened to
    • Tear the book in half, throw the first half away and repeat this process on the second half
  – If the person’s name is **before** the page you opened to
    • Tear the book in half, throw the second half away and repeat this process on the first half
• This is very hard on phone books, but you’ll find the name!
Binary Search
Binary Search

• We can use this to search sorted lists!
• To make our problem slightly easier, let’s make it the problem of “checking to see if something is in a sorted list”
  – For purposes of our example, if there’s no “middle” of the list, we’ll just look at the lower of the two possible indices
  – So if the list has 11 elements, the fifth one would be our middle
Binary Search

- Binary search is a problem that can be broken down into
  - Something simple (breaking a list in half)
  - A smaller version of the original problem (searching that half of the list)
- That means we can use ... recursion!
Exercise: Recursive Binary Search

• Write a recursive binary search!
• Remember to ask yourself:
  – What is our base case(s)?
  – What is the recursive step?
Exercise: Recursive Binary Search

• Write a recursive binary search!
• Remember to ask yourself:
  – What is our base case(s)?
  – What is the recursive step?

```python
def binarySearch(myList, item):
```

• A hint: in order to get the number at the middle of the list, use this line:
```
myList[len(myList) // 2]
```
def binarySearch(myList, item):
    if(len(myList) == 0):
        return False
    middle = len(myList) // 2
    if(myList[middle] == item):
        return True
    elif(myList[middle] < item):
        return binarySearch(myList[middle+1:], item)
    else:
        return binarySearch(myList[:middle], item)
Algorithm Run Time
Run Time for Search

• Say we have a list that does not contain what we’re looking for.
• How many things in the list does linear search have to look at for it to figure out the item’s not there for a list of 8 things?
  • 16 things?
  • 32 things?
Run Time for Search

• Say we have a list that does not contain what we’re looking for.

• What about for binary search?
  – How many things does it have to look at to figure out the item’s not there for a list of 8 things?
    – 16 things?
    – 32 things?

• Notice anything different?
Different Run Times

• These algorithms scale differently!
  – Linear search does work equal to the number of items in the list
  – Binary search does work equal to the $\log_2$ of the numbers in the list!

• A $\log_2(x)$ is basically asking “2 to what power equals x?”
  – This is the same as saying, “how many times must we divide x in half before we hit 1?”
Different Run Times

• As our list gets bigger and bigger, which of the search algorithms is faster?
  – Linear or binary search?

• How much faster is binary search?
Another Example
Sum of All Products

• Say we have a list, and we want find the sum of everything in that list multiplied by everything else in that list
  – So if the list is [1, 2, 3], we want to find the value of:
  – 1*1 + 1*2 + 1*3 + 2*1 + 2*2 + 2*3 + 3*1 + 3*2 + 3*3

• As an exercise, try writing this function!
  
  ```python
  def sumOfAllProducts(myList):
  ```
Exercise Solution

def sumOfAllProducts(myList):
    result = 0
    for item in myList:
        for item2 in myList:
            result += item * item2
    return result
Run Time for Sum of All Products

• How many multiplications does this have to do for a list of 8 things?
  • For 8 things, it does 64 multiplications
  – 16 things?
    • For 16 things, it does 256 multiplications
  – 32 things?
    • For 32 things, you do 1024 multiplications

• In general, if you give it a list of size $N$, you’ll have to do $N^2$ multiplications!
Asymptotic Analysis
Asymptotic Analysis

• For a list of size $N$, linear search does $N$ operations. So we say it is $O(N)$ (pronounced “big Oh of n”)

• For a list of size $N$, binary search does $\log(N)$ operations, so we say it is $O(\log(N))$

• For a list of size $N$, our sum of products function does $N^2$ operations, which means it is $O(N^2)$

• The function in the parentheses indicates how fast the algorithm scales
Example

• What is the big O of the following, given a list of size N:

```python
for i in myList:
    for j in myList:
        for k in myList:
            print(i*j*k)
```

• This will be $O(N^3)$
Any Other Questions?
General Announcements

• Lab 12 this week – last lab of the semester!

• Project 2 is out
  – Due by Tuesday, December 8th at 8:59:59 PM
  – Do NOT procrastinate!

• Next Class: Sorting
Announcements: Final Exam

• Final Exam will be held on Friday, December 11th from 3:30 to 5:30 PM

• Being held in three separate rooms
  • Section 1 (Gibson, MW @ 1) – CHEM 030
  • Section 7 (Dixon, TR @ 5:30) – CHEM 030
  • Section 13 (Dixon, TR @ 10) – CHEM 030
  • Section 19 (Morawski, MW @ 4) – PAHB 132
  • Section 25 (Gibson, TR @ 4) – PHYS 101

• Make sure you go to the correct room!
Announcements: Surveys

• Next class, we will be doing the in-class SCEQ (Student Course Evaluation Questionnaire) – This is an important metric for assessment

• The second survey is available and was announced on Blackboard – This is 1% of your grade, and is your chance to give feedback on your experience with the course