Lab Overview

- Exam review
- Review lab 8
- Prep for lab 9

Exam Prep

- Similar format to Exam 1
  - Written
  - Very short answer, short answer, coding
- Content is cumulative but focus on material since Exam 1
  - Defining classes is *not* on the exam

Lab Review

- Descriptions matter: your comments and variable names are an indicator of your level of understanding
  - Also important as a means of communicating between developers and/or users
- If variables or comments are incorrect, that is a good place to try to correct misunderstandings
- Goal: variable names help cement the abstract model of the problem in your mind

Adding New Code

- In general, don't change existing, working, well-tested functions unless told to do so
- Rather than changing encryptMessage to also handle "\n", it's better to make encryptFile adhere to the preconditions for encryptMessage
  - encryptMessage would need yet another check on every character to see if it's a "\n"
  - encryptFile knows that the "\n" is at the end of every line.
    - It can remove that easily and then add a "\n" back after encoding
- Ask yourself: what can best handle this functionality?

Difference btw File *Name* and *Object*

- File name is a `string`
- File object is a `file`
- Need the file **name** (a string) to create the file **object**

  - Need to remember data types because not explicit in Python
  - Use good variable names to help

File Reminders

- When you open a file, you should `close` the file

Partial Gymnastics Code

```
def main():
    scores = getScoresFromFile(filename)
    avgDiffScore = scores.pop(0)

    avgExecScore = calculateAverageExecScore(scores)
    …

def calculateAverageExecScore(listOfScores):
    listOfScores.sort()
    totalExecScore = 0
    for pos in range(1, len(listOfScores)-1):
        totalExecScore += listOfScores[x]
    average = totalExecScore/(len(listOfScores)-2)
    return average

…
```

pop returns and deletes first item in list

For space, no comments, partial solution

LAB 9 PREPARATION

Review: Defining our own classes

- How do we define a new class?
- What are defined methods like?
- What is the keyword that ***must*** be the first parameter of every defined method?
  - What does that parameter represent?
- What is the special method name for the constructor?
- What is the special method that helps with printing?
  - What is the API (input, returned) for that method?
- Where do we initialize the data that is needed to represent every object of a class? What is the term for that data?
  - How do we access that data?

Review: Defining our own classes

- How do we define a class?
  - `class` keyword
- What are defined methods like?
  - Functions
- What is the keyword that must be the first parameter of every defined method?
  - `self;` represents the object being acted upon
- What is the special method name for constructor?
  - `__init__`
- What is the special method that helps with printing?
  - `__str__(self)` – returns a string representation of the object
- Where do we initialize the data that is needed to represent every object of a class?
  - In the constructor.
  - Use `self._var` to represent that data. Can access that data in other methods as `self._var.`
  - Called *instance variables.*

Card Class (Incomplete)

Class Doc String

```
class Card:
    """ A class to represent a standard playing card.
    The ranks are ints: 2-10 for numbered cards, 11=Jack,
    12=Queen, 13=King, 14=Ace.
    The suits are strings: 'clubs', 'spades', 'hearts',
    'diamonds'."""

    def __init__(self, rank, suit):
        """Constructor for class Card takes int rank and
             string suit."""
        self._rank = rank
        self._suit = suit

    def getRank(self):
        "Returns the card's rank."
        return self._rank

    def getSuit(self):
        "Returns the card's suit."
        return self._suit
```

Method Doc String

Methods

Identify the ***instance variables***

- How do we use them in Card methods?

card.py

Card Class (Incomplete)

Class Doc String

```
class Card:
    """ A class to represent a standard playing card.
    The ranks are ints: 2-10 for numbered cards, 11=Jack,
    12=Queen, 13=King, 14=Ace.
    The suits are strings: 'clubs', 'spades', 'hearts',
    'diamonds'."""

    def __init__(self, rank, suit):
        """Constructor for class Card takes int rank and
             string suit."""
        self._rank = rank
        self._suit = suit

    def getRank(self):
        "Returns the card's rank."
        return self._rank

    def getSuit(self):
        "Returns the card's suit."
        return self._suit
```

Methods

Method Doc String

Identify the ***instance variables***
- How do we use them in Card methods?

Convention: instance variables are named beginning with _

card.py

Algorithm for Creating Classes

1. Identify need for a class
2. Identify state or attributes of a class/an object in that class
   - Determine how to model instance variables (their types)
   - Write the constructor (`__init__`)
3. Identify methods (i.e., functionality) the class should provide
   - `__str__` method
     - Test the `__str__` method
   - How will a user call those methods (parameters, return values)?
     - Develop API
4. Implement, test *one* method
   - Repeat until have complete API

Review: Testing our methods

- Can test similarly to how we tested functions

```
# create an object
c1 = Card(14, "spades")

# test the str method
test.testEqual( str(c1), "Ace of spades")

# test get rummy value
test.testEqual( c1.getRummyValue(), 15 )

# test the card color
test.testEqual( c1.getCardColor(), "black" )
```

Should test more than one object

Lab 9: Dealing with Real Data

- **Problem**: Determine most common first and last names at W&L
  - 4 data files, containing student names
    - Last names, female first names, male first names, all first names
    - 1 name per line
  - What data structure to use?
- Create a class to help with counting names
- Create output file used by another application
  - Common use of programming

Motivating using list's `sort` method with a *key*

- We may not want to sort a list of objects by the "standard" way to sort objects

- Consider sorting strings: How does Python sort/order strings usually?

Using list's `sort` method with a key

- We may not want to sort a list of objects by the "standard" way to sort objects
- Consider sorting strings: How does Python sort strings usually?
  - Alphabetically, upper-case first
- To alphabetize strings, sorting them by their lowercase value: `words.sort(key=str.lower)`

Method to call to do comparison

Using list's sort method with a key

```
words = ["Washington", "and", "Lee", "computer", "science"]
words.sort()

print("Words in Python str-standard sorted order:")
for word in words:
    print(word)
print()

print("Words in sorted order, ignoring upper and lower case:")

words.sort(key=str.lower)

for word in words:
    print(word)
```

Method is named as Classname.methodname

sort_ignore_case.py

Using list's sort method with a key

```
words = ["Washington", "and", "Lee", "computer", "science"]
words.sort()

print("Words in Python str-standard sorted order:")
for word in words:
    print(word)
print()

print("Words in sorted order, ignoring upper

words.sort(key=str.lower)

for word in words:
    print(word)
```

Method is named as Classname.methodna

```
Words in Python str-standard sorted order:
Lee
Washington
and
computer
science

Words in sorted order, ignoring upper and lower case:
and
computer
Lee
science
Washington
```

sort_ignore_case.py

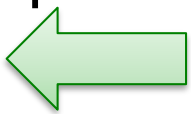
Lab Overview

(After warm up dictionary problem)

1. Implement partial solution using a dictionary to map the name to its count
   - handles basic set up of solution, including reading and processing file
2. Implement a class that packages the name (data) and its count together
   - Data and functionality given
   - Test the class
3. Implement Step 1 with objects of class you created in Step 2
   - Complete solution
4. Graph data generated from Step 3
5. Make web page with graphs

Graphing

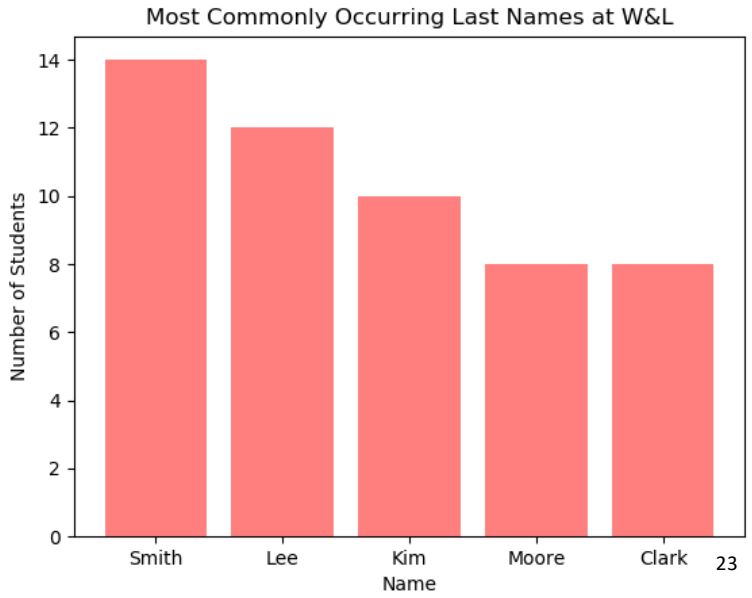
- I provide code that will create a bar chart using the `matplotlib` library
- You will need to provide the appropriate information to the Python code to generate the graph
  - You can either
    - Use the user interface (`generateFreqGraphs.py`)
    - Write code to directly call the `plotFrequencyData` function (`graphing_example.py`)



Graphing: Using the User Interface

```
$ python generateFreqGraphs.py
What is the name of your properly-formatted data file? data/lastnames.dat
How many results do you want to display? 5
What is the title of this graph? Most Commonly Occurring Last Names at W&L
What is the y-axis label of this graph? Number of Occurrences
What is the filename you want for the graph? data/lastnames.png
['Smith', '14']
['Lee', '12']
['Kim', '10']
['Moore', '8']
['Clark', '8']
```

Generates Graph:



Graphing: Using Function Calls

```
from generateFreqGraphs import *

DATADIR="data/"

nameLabels, dataToPlot = processDataFile(DATADIR + \
          "data/lastnames.dat", 6)

plot = plotFrequencyData(nameLabels, dataToPlot, \
     "Most Commonly Occurring Last Names at W&L", \
     "Number of Students", DATADIR + "lastnames.png")
```

Recommended way

graphing_example.py

Overview

(After warm up dictionary problem)

1. Implement partial solution using a dictionary to map the name to its count
   - handles basic set up of solution, including reading and processing file
2. Implement a class that packages the name (data) and its count together
   - Data and functionality given
   - Test the class
3. Implement Step 1 with objects of class you created in Step 2
   - Complete solution
4. Graph data generated from Step 3
5. Make web page with graphs

FNL


CLEAR MINDS,
FULL HEARTS,
CAN'T LOSE!
- FRIDAY NIGHT LIGHTS
COMPUTATIONAL THINKING