CompSci 101
Fall 2021

Lecture 20

Reminders

- **NO CLASS TUESDAY! (TRAVEL SAFELY!)**

- **Assignments**
  - APT 7 due TOMORROW (11/19)
  - APT 8 live
  - Assign 6 and 7 live
    - GRACE PERIOD ENDS DEC 3! NOT ACCEPTED AFTER THIS!
- **Assessment sent via Learning Innovation**
  - 80% response rate by 11/30→Extra credit

- **Spring 2022 UTA applications open**
  - See Ed announcement

Key instructions

- **Input ✓**
- **Output ✓**
- **Assignments* ✓**
- **Math/Logic ✓**
- **Conditionals✓**
- **Repetition ✓**

not listed in book

Python Data Types

- **int, float, bool ✓**
- **Collections**
  - Strings ✓
  - Lists ✓
  - Tuples ✓
  - Sets ✓
  - Dictionaries ✓

PFTD

- **Exceptions**
- **Recommender**
  - Recommendations big picture
  - Assignment big picture
  - Simple recommendation example
  - Actual recommendation assignment

KISS Principle

- **Think of the non-computing context for any word/terms**
- **KISS model**
  - Work smarter, not harder!!
- **“Good programmers are simply good designers.”**
  - *-Dr. Washington*
- **Design first and always!**
- **Importance of reusability**
- ***USE PyCharm/PythonTutor IF YOU HAVE QUESTIONS!***

People to Know: Andrea Delgado-Olson

- BS/MS (Mills College)
- Native American Women in Computing
  - Founder and Chair
- Program Manager
  - GHC Communities and Systers
- Created Udacity course (Android Basics Nanodegree for Multiscreen Apps) in her native language (Miwok).
- Ione Miwok

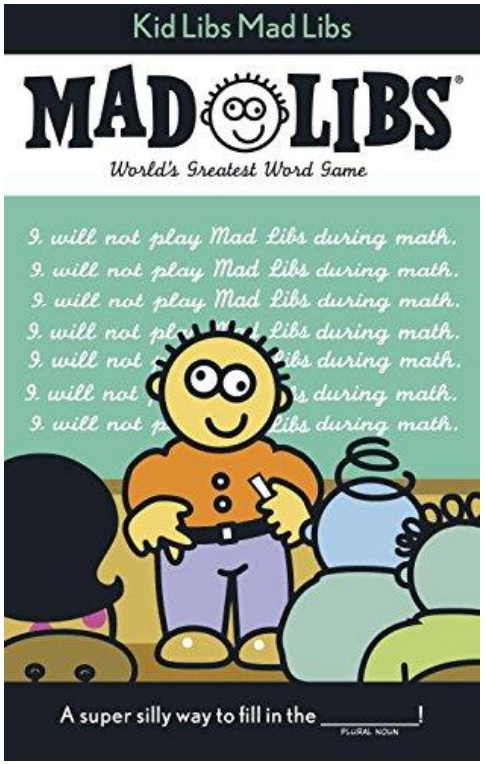
Why use modules?

- **Easier to organize code**
- **Easier to reuse code**
- **Easier to change code**
  - As long as the “what” is the same, the “how” can change
    - Ex: sorted(…), one function many sorting algorithms

In laterLab, Modules for Creating

- **“~~MadLibs~~” → Tag-a-Story**
  - User chooses template
  - Computer fills everything in

```
In lecture I saw a <color> <noun>
For lunch I had a <adjective> <food>
The day ended with seeing a <animal>
<verb> in <place>
```



From <noun> to story

In lecture I saw a <color> <noun>

For lunch I had a <adjective> <food>

The day ended with seeing a <animal> <verb> in <place>

In lecture I saw a magenta house

For lunch I had a luminous hummus

The day ended with seeing a cow sleep in Mombasa

Demo

Let's create/modify a story

- **Choose a template or make a new one**
  - We'll choose lecturetemplate.txt first
- **Add a new category/replacement**
  - We'll choose number and list some choices
- **Run the program and test our modifications**
  - Randomized, hard to test, but doable

Main Parts for tag-a-story

- **Put everything together, the template and words**
  - Storyline.py
- **Loading and handling user choosing templates**
  - TemplateChooser.py
- **Loading and picking the word for a given tag**
  - Replacements.py

Main Parts for tag-a-story

- **Put everything together, the template and words**
  - Storyline.py
- **Loading and handling user choosing templates**
  - TemplateChooser.py
- **Loading and picking the word for a given tag**
  - Replacements.py

Creating a story

- **Main steps in Storyline.py**
  - Get template – use a module
  - Go through template
    - Get words for a tag – use a module
    - Replace tag with word

- **Using modules**
  - Assume they work
  - Only care ***what*** they do, not ***how*** (abstraction!)

- ***If creating modules, you WOULD need to test them to make sure they work correctly.***

Modules in Action: makeStory() is in Storyline.py

- **How can we access TemplateChooser functions?**
  - import and access as shown

```
41  def makeStory():
42      """
43      let user make a choice of
44      available templates and print
45      the story from the chosen template
46      """
47      lines = TemplateChooser.getTemplateLines("templates")
48      st = linesToStory(lines)
49      print(st)
```

Modules in Action: makeStory() is in Storyline.py

- **How can we access TemplateChooser functions?**
  - import and access as shown

Another module (file)

```
41  def makeStory():
42      """
43      let user make a choice of
44      available templates and print
45      the story from the chosen template
46      """
47      lines = TemplateChooser.getTemplateLines("templates")
48      st = linesToStory(lines)
49      print(st)
```

Modules in Action: makeStory() is in Storyline.py

- **How can we access TemplateChooser functions?**
  - import and access as shown

A function in the file: TemplateChooser.py

```
41  def makeStory():
42      """
43      let user make a choice of
44      available templates and print
45      the story from the chosen template
46      """
47      lines = TemplateChooser.getTemplateLines("templates")
48      st = linesToStory(lines)
49      print(st)
```

Understanding Code/Module

doWord is in Storyline.py

- **What does getReplacement do?**
  - How does getReplacement do it?

```
10  def doWord(word):
11      """
12      word is a string
13      if word is <tag>, find replacement
14      and return it. Else return word
15      """
16      start = word.find("<")
17      if start != -1:
18          end = word.find(">")
19          tag = word[start+1:end]
20
21          rep = Replacements.getReplacement(tag)
22          return rep
23      return word
```

Understanding Code/Module

doWord is in Storyline.py

- **What does getReplacement do?**
  - How does getReplacement do it?

```
10  def doWord(word):
11      """
12      word is a string
13      if word is <tag>, find replacement
14      and return it. Else return word
15      """
16      start = word.find("<")
17      if start != -1:
18          end = word.find(">")
19          tag = word[start+1:end]
20
21          rep = Replacements.getReplacement(tag)
22          return rep
23      return word
```

Another module (file)

Understanding Code/Module

doWord is in Storyline.py

- **What does getReplacement do?**
  - How does getReplacement do it?

```
10  def doWord(word):
11      """
12      word is a string
13      if word is <tag>, find replacement
14      and return it. Else return word
15      """
16      start = word.find("<")
17      if start != -1:
18          end = word.find(">")
19          tag = word[start+1:end]
20
21          rep = Replacements.getReplacement(tag)
22          return rep
23      return word
```

A function in the file: Replacements.py

The other module's "what"

- **Get template**
  - `TemplateChooser.getTemplateLines(DIR)`
  - What:
    - From the templates in the directory DIR (type: str)
    - Return a list of strings, where each element is a line from one of the templates in DIR
- **Word for a tag**
  - `Replacements.getReplacement(TAG)`
  - What:
    - Return a random word that matches TAG (type: str)

Main Parts for tag-a-story

- **Put everything together, the template and words**
  - Storyline.py

- **Loading and handling user choosing templates**
  - TemplateChooser.py

- **Loading and picking the word for a given tag**
  - Replacements.py

TemplateChooser.py Steps

- **List all templates in the folder**
- **Get user input that chooses one**
- **Load that template**
- **Return as list of strings**

TemplateChooser.py Steps

- **List all templates in the folder**
  - pathlib Library
- **Get user input that chooses one**
  - Handle bad input → try…except
- **Load that template**
  - Open file, .readlines()
- **Return as list of strings**

These Steps in Code getTemplateLines in TemplateChooser.py

- **Read directory of templates, convert to dictionary**
  - Let user choose one, open and return it

```
59  def getTemplateLines(dirname):
60      """
61      dirname is a string that's the name of a folder
62      Prompt user for files in folder, allow user
63      to choose, and return the lines read from file
64      """
65      d = dirToDictionary(dirname)
66      lines = chooseOne(d)
67      return lines
```

Creating User Menu dirToDictionary in TemplateChooser.py

- **What does this function return? What type?**

```
11  def dirToDictionary(dirname):
12      """..."""
18      d = {}
19      index = 0
20      for one in pathlib.Path(dirname).iterdir():
21          d[index] = one
22          # print(type(one))
23          index += 1
24      return d
```

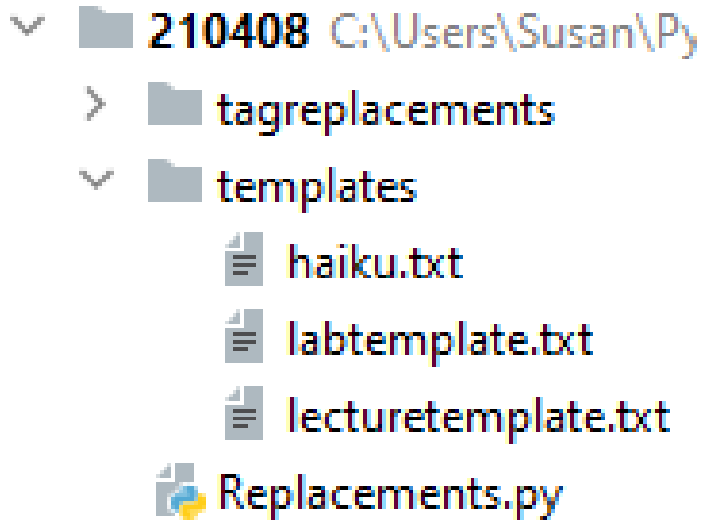
Creating User Menu dirToDictionary in TemplateChooser.py

- **What does this function return? What type?**

```
11  def dirToDictionary(dirname):
12      """..."""
18      d = {}
19      index = 0
20      for one in pathlib.Path(dirname).iterdir():
21          d[index] = one
22          # print(type(one))
23          index += 1
24      return d
```

d is:
0 -> haiku.txt
1 -> labtemplate.txt
2 -> lecturetemplate.txt

Folder in Pycharm



Output:

```
C:\Users\Susan\AppData\Lo
0   haiku.txt
1   labtemplate.txt
2   lecturetemplate.txt
------
choose one> 0
the slimy bathtub
reminded them of Africa
chartreuse squeaky brown
```

`pathlib` Library

- **Path: “`rodger/Pycharm/cps101/lab11/temp/haiku.txt`”**

- **The `pathlib` library is more recent/Python3**
  - Simpler, easier to use than functions from **`os`**

- **Handles domain specifics!**
  - Doesn’t matter if on Windows, Mac, etc.
  - We worry about the ***what***, it handles the ***how***

`pathlib` Library cont.

- **Path: “`rodger/Pycharm/cps101/lab11/temp/haiku.txt`”**

- **`pathlib.Path(DIR).iterdir()`**
  - Returns iterable of Path objects representing each “thing” in the directory DIR

- **Path object’s .parts – tuple of strings, each element is a piece of a filename’s path**
  - `(‘rodger’, ‘Pycharm’, ‘cps101’,’lab11’, ‘temp’, ‘haiku.txt’)`

Understanding the Unknown chooseOne in TemplateChooser.py

- **We will return to this, but analyze parts now**
  - What's familiar? What's not familiar …

```
39  def chooseOne(d):
40      """..."""
46      while True:
47          for key in sorted(d.keys()):
48              print("%d\t%s" % (key, d[key].parts[-1]))
49          print("------")
50          st = input("choose one> ")
51          try:
52              val = int(st)
53              if 0 <= val and val < len(d):
54                  return reader(d[val])
55          except ValueError:
56              print("please enter a number")
```

Python exceptions

- **What should you do if you prompt user for a number and they enter "one"**
  - Test to see if it has digits?

- **Use exceptions with `try:` and `except:`**
  - See code in function **`chooseOne`** from ***TemplateChooser.py***

Handling Exceptions

- **What happens: `x = int("123abc")`**

```
46          st = input("choose one> ")
47          try:
48              val = int(st)
49              if 0 <= val and val < len(d):
50                  return reader(d[val])
51          except ValueError:
52              print("please enter a number")
```

When and What's in CompSci 101

- **Problem to solve**
  - Use 7 steps
  - Step 5: How do you translate algorithm to code?
    - What do you use to solve it?
    - When do you use it?

What are the “what’s”?

- **Data Structures: list, set, dictionary, tuple**
- **Loops and iterables: from for to while to iterdir()**
- **Other:**
  - List comprehensions
  - Parallel lists
  - Lambda
  - If…if…if
  - If…elif…else

Quick When's and What's for 101

- **Whichever makes more sense to you:**
  - Parallel lists vs dictionaries
  - If…if…if vs if…elif…else
  - List comprehension vs for loop
  - Tuples vs Lists
    - If you want to prevent mutation -> tuples
  - Need single line function
    - Lambda vs create normal helper function

APT – Sorted Freqs

APT SortedFreqs

Problem Statement

Specification

```
filename: SortedFreqs.py

def freqs(data):
    """
    return list of int values corresponding
    to frequencies of strings in data, a list
    of strings
    """
```

The frequency with which data occurs is sometimes an important statistic. In this problem you'll determine how frequently strings occur and return a list representing the frequencies of each different/unique string. The list returned contains as many frequencies as there are unique strings. The returned frequencies represent an alphabetic/lexicographic ordering of the unique words, so the first frequency is how many times the alphabetically first word occurs and the last frequency is the number of times the alphabetically last word occurs.

Consider these strings (quotes for clarity, they're not part of the strings).

```
["apple", "pear", "cherry", "apple", "cherry", "pear", "apple", "banana"]
```

The list returned is `[3,1,2,2]` since the alphabetically first word is `"apple"` which occurs 3 times; the second word alphabetically is `"banana"` which occurs once, and the other words each occur twice.

What's the best way to …

- **SortedFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortedfreqs.html

- **Count how many times each string occurs**
  - Create d = {}, iterate over list updating values
  - Use data.count(w) for each w

What's the best way to …

- **SortedFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortedfreqs.html

- **Count how many times each string occurs**
  - Create d = {}, iterate over list updating values
  - Use data.count(w) for each w
    - Wait, that looks like …

```
6   def freqs(data):
7       return [data.count(d) for d in sorted(set(data))]
```

APT: SortByFreqs

APT SortByFreqs

Problem Statement

Specification

```
filename: SortByFreqs.py

def sort(data):
    """
    return list of strings based on
    the list of strings in parameter data
    """
```

The frequency with which data occurs is sometimes an important statistic. In this problem you are given a list of strings and must determine how frequently the strings occur. Return a list of strings that is sorted (ordered) by frequency. The first element of the returned list is the most frequently occurring string, the last element is the least frequently occurring. Ties are broken by listing strings in lexicographic/alphabetical order. The returned list contains one occurrence of each unique string from the list parameter.

Consider these strings (quotes for clarity, they're not part of the strings).

```
["apple", "pear", "cherry", "apple", "pear", "apple", "banana"]
```

The list returned is:

```
[ "apple", "pear", "banana", "cherry" ]
```

since the most frequently occurring string is `"apple"` which occurs 3 times; the string `"pear"` occurs twice and the other strings each occur once so they are returned in alphabetical order.

Wait, wait, but what's …

- **SortByFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html

- **Sort by # occurrences high to low**
  - Tuples with count/lambda and reverse=True?
  - Break ties in alphabetical order: two passes

Wait, wait, but what's …

- **SortByFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html

- **Sort by # occurrences high to low**
  - Tuples with count/lambda and reverse=True?
  - Break ties in alphabetical order: two passes

```
6  def sort(data):
7      tups = [(data.count(t),t) for t in sorted(set(data))]
8      result = [t[1] for t in sorted(tups,key=lambda x : x[0],reverse=True)]
9      return result
```

SortByFreqs Example

- **SortByFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html

```
6   def sort(data):
7       tups = [(data.count(t),t) for t in sorted(set(data))]
8       result = [t[1] for t in sorted(tups,key=lambda x : x[0],reverse=True)]
9       return result
```

SortByFreqs Example

- **SortByFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html

```
6   def sort(data):
7       tups = [(data.count(t),t) for t in sorted(set(data))]
8       result = [t[1] for t in sorted(tups,key=lambda x : x[0],reverse=True)]
9       return result
```

data = ["apple", "pear", "cherry", "apple", "pear", "apple", "banana“]

tups = [(3, “apple”),(1, “banana”),(1,”cherry”),(2,”pear”)]

sorted(…) line 8 = [(3,”apple”),(2,”pear”),(1, “banana”),(1,”cherry”)]

result = [“apple”,”pear”,”banana”,”cherry”]

PRINT A LOT!

Activity 1: https://bit.ly/101f21-11-18-1

Reminders

- **Work smarter, not harder**
- **Design first**
- **Get smaller parts working, then build on it**
- **Try to identify where you are stuck**
  - Identify resources to help solve problem
- **Leverage your design and PythonTutor to understand program flow of control**
  - http://pythontutor.com