Compsci 101
Modules, How Dictionaries Work

Susan Rodger
April 7, 2022

V is for …

- Viral Video
  - Husky Dog sings with iPAD – 18 million views
    - https://www.youtube.com/watch?v=Mk4bmK-acEM
- Virtual Memory
  - It is and is not there!
- Virtual Reality
  - Augmenting IRL

The Power of Collaboration:
Ge Wang, Duke Prof. at Stanford

- Duke 2000: Music and Computer Science
  - http://www.youtube.com/watch?v=ADEHmkL3HBq
- About Design in Compsci 308

  *Our investment into a huge and meticulous design process was a huge factor in making later progress. 35000+ lines of code / design / documentation gave us a project we were all very happy and proud to be a part of.*

Announcements

- APT-7 due TODAY!
- APT-8 out, due Thursday, Apr 14
- Assign 6 Recommender, due Apr 19
  - One grace day, NO LATE DAYS, must be in Apr 20
- APT Quiz 2 – 11:30am today thru Sunday, April 10
  - Two Parts, Start on Sakai
  - Rules were sent to you, must be your own work!
- Exam 4 – Tues, April 12, in person
  - See study materials on calendar page on 4/12 date
PFTD

- Collaboration and Creativity
  - The power of working together with code
- Review modules and exceptions
  - Concepts used in Lab 11, leveraging creativity
- How dictionaries are so fast
- Exam review

Why use modules?

- Module – Python file (.py file)
- Can have several modules work together
- 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 magenta house
For lunch I had a luminous hummus
The day ended with seeing a cow sleep in Mombasa
Demo

• Run storyline.py
• Show Haiku's
• Show Lecture template
• Make modifications

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
Creating a story

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

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

---

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

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

```python
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: linesToStory() is in Storyline.py

- **We call doWord() – does replacements for words**

```python
27 def linesToStory(lines):
28     
29     lines is a list of strings,
30     each a line from a template file
31     Return a string based on substituting
32     for each <tag> in each line
33     
34     story = ""
35     for line in lines:
36         
37         for word in line.split():
38             st = ""
39             
40             return story
```

---

Understanding Code/Module doWord is in Storyline.py

- **What does getReplacement do in Storyline.py**
  - How does getReplacement do it?
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

Another module TemplateChooser.py

- 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)

Where is it called from?

- In module Storyline.py, function makestory

  lines = TemplateChooser.getTemplateLines("templates")

- Where templates is a folder with three templates:

  ![templates]

  - haiku.txt
  - labtemplate.txt
  - lecturetemplate.txt

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
def getTemplateLines(dirname):
    """
    dirname is a string that's the name of a folder
    Prompt user for files in folder, allow user
    to choose, and return the lines read from file
    """
    d = dirToDictionary(dirname)
    lines = chooseOne(d)
    return lines

Creating User Menu
def dirToDictionary(dirname):
    d = {}
    index = 0
    for one in pathlib.Path(dirname).iterdir():
        d[index] = one
        # print(type(one))
        index += 1
    return d
Folder in Pycharm

- C:\Users\Susan\Pycharm\tagreplacements
- templates
  - haiku.txt
  - labtemplate.txt
  - lecturtemplate.txt
  - Replacements.py

Output:

```
C:\Users\Susan\AppData\Local\...
0  haiku.txt
1  labtemplate.txt
2  lecturtemplate.txt
------
choose one> 
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.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

pathlib Library cont.

Understanding the Unknown

chooseOne in TemplateChooser.py

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

```
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")
```

Handling Exceptions

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

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

WOTO-1 Modules

How do Dictionaries work so fast?

- How are they implemented?
Simple Example
Want a mapping of Soc Sec Num to Names

- Duke’s CS Student Union wants to be able to quickly find out info about its members. Also add, delete and update members. Doesn't need members sorted.
  - 267-89-5431  John Smith
  - 703-25-6141  Ademola Olayinka
  - 319-86-2115  Betty Harris
  - 476-82-5120  Rose Black

- Dictionary d – SSN to names
  - d['267-89-5431'] = ‘John Smith’
  - How does it find ‘John Smith’ so fast?

Dictionary d(SSN) = (SSN, name)

- We actually would map the SSN to the tuple of (SSN, name).
- That is a lot to display on a slide, so we will just show SSN to name
- But remember name is really a tuple of (SSN, name)

Simple Example
Let's look under the hood.
How are dictionaries implemented?

- Dictionaries implemented with a list, in a clever way
- How do we put something into the list fast?
  - d['267-89-5431'] = ‘John Smith’
- How do we find it in the list quickly?
  - List size is 11 – from 0 to 10
  - d['267-89-5431'] calculates index location in list of where to put this tuple (SSN,name)
  - Use a function to calculate where to store ‘John Smith
    - H(ssn) = (last 2 digits of ssn) mod 11
    - Called a Hash function

Have a list of size 11 from 0 to 10

- Insert these into the list
- Insert as (key, value) tuple
  (267-89-5431, John Smith)
(in example, only showing name)
When does this work well?

- When there are few collisions
- You have to deal with collisions
- Use a list large enough to spread out your data

Another way: Use a list of lists

- Insert these into the list
- Insert as (key, value) tuple (267-89-5431, John Smith) (in example, only showing name)

| H(267-89-5431) = 31 %11 = 9 |
| John Smith |
| H(703-25-6141) = 41%11 = 8 |
| Ademola Olayinka |
| H(319-86-2115) = 15 %11 = 4 |
| Betty Harris |
| H(476-82-5120) = 20%11 = 9 |
| Rose Black |

WOTO-2 How Dictionaries Work

Review for Exam 4
Problem 4 Fall 2014 Old Tests

- A programming contest between colleges
- There are problems to solve each has a letter: Problem A through Problem J
- Submit a program for a problem – it is correct or not
- Submit it again if it is not correct.
- Score is total time for problems solved with 20 minute penalty for each wrong submission that was solved eventually!
- Winner is solves most problems – Tie breaker (lowest score)
Review for Exam 4
Problem 4 Fall 2014 Old Tests

- Each entry is: 1) school, 2) name of problem, 3) time to solve in minutes, 4) correct or not
- Examples:
  - ['UNC', 'A', '20', 'reject']
  - ['Duke', 'A', '26', 'correct']

Problem 4 Fall 2014 Old Tests
Just look at Duke’s submissions

Duke score:

- ['Duke', 'A', '26', 'correct'],
- ['Duke', 'E', '82', 'reject'],
- ['Duke', 'D', '200', 'correct'],
- ['Duke', 'E', '210', 'correct'],

Problem 4 Fall 2014 Old tests
data is list of lists of submissions

data = [
  ['UNC', 'A', '20', 'reject'],
  ['Duke', 'A', '26', 'correct'],
  ['UNC', 'A', '33', 'reject'],
  ['ECU', 'A', '34', 'correct'],
  ['Elon', 'A', '34', 'correct'],
  ['USC', 'G', '44', 'reject'],
  ['UNC', 'A', '45', 'correct'],
  ['NCSU', 'B', '60', 'reject'],
  ['USC', 'C', '72', 'reject'],
  ['Duke', 'E', '82', 'reject'],
  ['USC', 'C', '90', 'correct'],
  ['NCSU', 'A', '33', 'reject'],
  ['Duke', 'A', '45', 'correct'],
  ['USC', 'D', '200', 'correct'],
  ['ECU', 'F', '202', 'reject'],
  ['Duke', 'D', '200', 'correct'],
  ['Duke', 'E', '210', 'correct'],
  ['UNC', 'E', '212', 'reject'],
  ['USC', 'G', '220', 'reject'],
  ['NCSU', 'D', '222', 'correct'],
  ['Elon', 'H', '225', 'correct'],
  ['NCSU', 'H', '230', 'reject']]

Write function listOfSchools(data)

- returns sorted unique list of schools that submitted a program whether correct or not
- From data should return:

  ['Duke', 'ECU', 'Elon', 'NCSU', 'UNC', 'USC'].

Unique schools (use sets) Returns list (must convert set back to list)

Note: sorted
Write function `listOfSchools(data)`

def listOfSchools(data):
    setSchools = set([])
    for item in data:
        setSchools.add(item[0])
    alist = list(setSchools)
    return sorted(alist)

Problem 4 Fall 2014 Old tests

data is list of lists of submissions

data = [
    ['UNC', 'A', '20', 'reject'],
    ['Duke', 'A', '26', 'correct'],
    ['UNC', 'A', '33', 'reject'],
    ['ECU', 'A', '34', 'correct'],
    ['Elon', 'A', '34', 'correct'],
    ['USC', 'G', '44', 'reject'],
    ['UNC', 'A', '45', 'correct'],
    ['NCSU', 'B', '60', 'reject'],
    ['USC', 'C', '72', 'reject'],
    ['Duke', 'E', '82', 'reject'],
    ['USC', 'C', '90', 'correct'],
    ['UNC', 'B', '98', 'reject'],
    ['NCSU', 'B', '103', 'correct'],
    ['NCSU', 'A', '115', 'correct'],
    ['USC', 'A', '116', 'correct'],
    ['ECU', 'F', '202', 'reject'],
    ['Duke', 'D', '200', 'correct'],
    ['Duke', 'E', '210', 'correct'],
    ['UNC', 'B', '212', 'reject'],
    ['USC', 'G', '220', 'reject'],
    ['NCSU', 'D', '222', 'correct'],
    ['Elon', 'H', '225', 'correct'],
    ['NCSU', 'H', '230', 'reject']]

Write function `problemsAttempted(data)`

- Returns list of problems attempted
- **Would return list:**
  - ['A', 'C', 'B', 'E', 'D', 'G', 'F', 'H']
  - Note doesn’t say anything about the order but implies one of each.
Write function
\[
\text{problemsNotAttempted}(\text{problems, data})
\]
- problems is a list of all possible problems
- Returns a list of the problems not attempted

```
def problemsNotAttempted(problems, data):
    data = [  
        ['UNC', 'A', '20', 'reject'],  
        ['Duke', 'A', '26', 'correct'],  
        ['UNC', 'A', '33', 'reject'],  
        ['ECU', 'A', '34', 'correct'],  
        ['Elon', 'A', '34', 'correct'],  
        ['USC', 'G', '44', 'reject'],  
        ['UNC', 'A', '45', 'correct'],  
        ['NCSU', 'B', '60', 'reject'],  
        ['USC', 'C', '72', 'reject'],  
        ['Duke', 'E', '82', 'reject'],  
        ['USC', 'C', '90', 'correct'],  
        ['UNC', 'B', '98', 'reject'],  
        ['NCSU', 'B', '103', 'correct'],  
        ['NCSU', 'A', '115', 'correct'],  
        ['USC', 'A', '116', 'correct'],  
        ['ECU', 'F', '202', 'reject'],  
        ['Duke', 'D', '200', 'correct'],  
        ['Duke', 'E', '210', 'correct'],  
        ['UNC', 'B', '212', 'reject'],  
        ['USC', 'G', '220', 'reject'],  
        ['NCSU', 'D', '222', 'correct'],  
        ['Elon', 'H', '225', 'correct'],  
        ['NCSU', 'H', '230', 'reject']  
    ]
    attempted = problemsAttempted(data)  
    setProbs = set(problems)  
    setAttempted = set(attempted)  
    setNotAttempted = setProbs - setAttempted  
    return list(setNotAttempted)
```
Write function `dictProblemstoSchoolsSolved(data)`

- Returns a dictionary of letters for problems mapped to list of schools that solved that problem
  - ‘B’ mapped to ['NCSU']
  - ‘A’ mapped to ['Duke', 'ECU', 'Elon', 'UNC', 'NCSU', 'USC']
  - ‘D’ mapped to ['Duke', 'NCSU']
  - Etc

```python
def dictProblemsToSchoolsSolved(data):
    d = {}
    for item in data:
        if item[3] == 'correct':
            if item[1] in d:
                d[item[1]].append(item[0])
            else:
                d[item[1]] = [item[0]]
    return d
```

Problem 4 Fall 2014 Old tests

data is list of lists of submissions

```python
data = [
    ['UNC', 'A', '20', 'reject'],
    ['Duke', 'A', '26', 'correct'],
    ['UNC', 'A', '33', 'reject'],
    ['ECU', 'A', '34', 'correct'],
    ['Elon', 'A', '34', 'correct'],
    ['USC', 'G', '44', 'reject'],
    ['UNC', 'A', '45', 'correct'],
    ['UNC', 'B', '60', 'reject'],
    ['USC', 'C', '72', 'reject'],
    ['Duke', 'E', '82', 'reject'],
    ['USC', 'C', '90', 'correct'],
    ['UNC', 'B', '98', 'reject'],
    ['NCSU', 'B', '103', 'correct'],
    ['NCSU', 'A', '115', 'correct'],
    ['USC', 'A', '116', 'correct'],
    ['ECU', 'F', '202', 'reject'],
    ['Duke', 'D', '200', 'correct'],
    ['Duke', 'E', '210', 'correct'],
    ['UNC', 'B', '212', 'reject'],
    ['USC', 'G', '220', 'reject'],
    ['NCSU', 'D', '222', 'correct'],
    ['Elon', 'H', '225', 'correct'],
    ['NCSU', 'H', '230', 'reject']
]```

Write function `dictSchoolsToNumSubmissions(data)`

- Returns a dictionary of schools mapped to the number of submissions they had (rejected or correct)
  - ‘Duke’ mapped to 4
  - ‘UNC’ mapped to 5
  - Etc

```python
def dictSchoolsToNumSubmissions(data):
    d = {}
```
Write function
dictSchoolsToNumSubmissions(data)

```python
def dictSchoolsToNumSubmissions(data):
    d = {}
    for item in data:
        if item[0] in d:
            d[item[0]] += 1
        else:
            d[item[0]] = 1
    return d
```

WOTO-3 Solving problems