

Objectives

- Deal or No Deal
- Dictionaries

Mar 14, 2011

Sprenkle - CSCI1111

1

Review

- How can we iterate through a list?
- True or False: If we modify a *list* variable inside a function, it is modified outside of the function
- True or False: If we modify an *integer* variable inside a function, it is modified outside of the function

Mar 14, 2011

Sprenkle - CSCI1111

2

Passing Lists as Parameters

- Lists are **mutable**
- Can be modified inside of functions
 - Modifications seen outside of functions
- Example from Friday:
 - `descendSort3Nums(list3)`

Mar 14, 2011

Sprenkle - CSCI1111

3

Modifying Lists Passed as Parameters

```
def descendSort3Nums(list3):  
    if list3[1] > list3[0]:  
        # swap 'em  
        tmp = list3[0]  
        list3[0] = list3[1]  
        list3[1] = tmp  
  
    if list3[2] > list3[1]:  
        tmp = list3[1]  
        list3[1] = list3[2]  
        list3[2] = tmp  
  
    if list3[1] > list3[0]:  
        tmp = list3[0]  
        list3[0] = list3[1]  
        list3[1] = tmp
```

```
def main():  
    myList = [1,2,3]  
    descendSort3Nums(list)  
    print myList
```

myList gets modified by function

Note: Function does not return anything. Simply modifies the list3 parameter.

Mar 14, 2011

Sprenkle - CSCI1111

4

Lab 8: Deal or No Deal Overview

- Have 26 cases with various amounts of money
 - Amounts are known
- Player selects a case (hope has the big jackpot)
- In each round, player opens up cases
 - Reveals amounts that are not in the case they chose
- Banker makes an offer to buy the case
- Player decides if want to take the deal
 - Is the offer more than what is in the case?
 - Make decision based on amounts that haven't been opened yet
- Game ends when only one more case to open (two amounts on board) or player takes the deal.

Mar 14, 2011

Sprenkle - CSCI1111

5

Implementing Deal or No Deal

- Given: partial solution in code
 - Complete `main()` function, some additional functions
- Your job:
 - Read, understand given code
 - Fill in the functions for a complete solution

Mar 14, 2011

Sprenkle - CSCI1111

6

Modeling *Deal or No Deal*

- Cases, numbered 0 to 25
 - How can we represent that a case has been opened?
 - Have dollar amounts in them

1000000	1000	5		750000	value
0	1	2	...	25	case/ position

- Board
 - Which dollar amounts have been chosen, which are still in play

.01	1	5		1000000	value
0	1	2	...	25	position

Mar 14, 2011

Sprenkle - CSCI1111

7

Modeling *Deal or No Deal*

CHOSEN = -1
means case opened:
Don't display on board,
Don't allow user to select again

- Cases, numbered 0 to 25
 - Have dollar amounts in them

1000000	1000	5		CHOSEN	value
0	1	2	...	25	case/ position

- Board
 - Which dollar amounts have been chosen, which are still in play

.01	CHOSEN	5		1000000	value
0	1	2	...	25	position

Mar 14, 2011

Sprenkle - CSCI1111

8

Functionality

- Read in values contained in cases from a file
 - What data type should these numbers be?
- Display remaining cases
 - Print four to a row
- Display remaining amounts on board
 - Left column is smaller amounts
- Have user select from remaining cases
 - Make sure choice is valid

Mar 14, 2011

Sprenkle - CSCI1111

9

Where is Documentation Coming From?

- Comes from the code itself in “**doc strings**”
 - i.e., “documentation strings”
- Doc strings are simply strings *after* the function header
 - Typically use triple-quoted strings because documentation goes across several lines

```
def printVerse(animal, sound):
    """ prints a verse of Old
    MacDonald, filling in the strings for
    animal and sound """
```

Mar 14, 2011

Sprenkle - CSCI1111

10

How to print remaining cases?

- Cases, numbered 0 to 25
 - Have dollar amounts in them

1000000	1000	5		CHOSEN	value
0	1	2	...	25	case/ position

- Board
 - Which dollar amounts have been chosen, which are still in play

.01	CHOSEN	1000		-1	value
0	1	2	...	25	position

Mar 14, 2011

Sprenkle - CSCI1111

11

How Does **in** Work for Lists?

- Example: `guess in prevGuesses`, where `prevGuesses` is a list object
 - For each element in list, checks if element equals (==) `guess`
- In the worst case, how many elements does **in** have to check?
 - How could we improve the search?

Mar 14, 2011

Sprenkle - CSCI1111

12

Faster Lookups

- If I wanted to know the Registrar's phone number, ...
 - Would I search through an alphabetized list of phone numbers?
 - No, I would look up the Registrar and find the phone number **associated** with the Registrar
- This type of data structure is known as a **dictionary** in Python
 - Maps a **key** to a **value**
 - Phone book's key: "Registrar", value: phone number

Mar 14, 2011

Sprenkle - CSCI111

13

Examples of Dictionaries

Dictionary	Keys	Values
Dictionary		
Textbook's index		
Cookbook		
URL (Uniform Resource Locator)		

- Any other things we've done/used in class?

Mar 14, 2011

Sprenkle - CSCI111

14

Examples of Dictionaries

Dictionary	Keys	Values
Dictionary	Word	Definition
Textbook's index	Keyword	Page number
Cookbook	Food type	Recipes
URL (Uniform Resource Locator)	URL	Web page

- Any other things we've done/used in class?

Mar 14, 2011

Sprenkle - CSCI111

15

Examples of Dictionaries

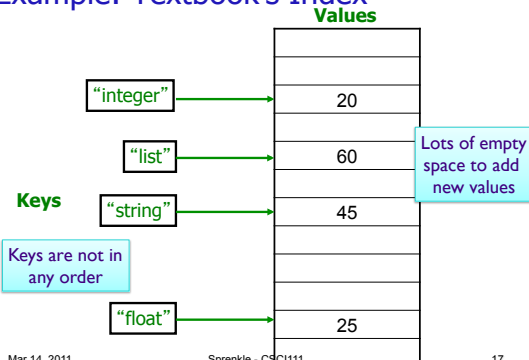
- Real-world:
 - Dictionary
 - Textbook's index
 - Cookbook
 - URL (Uniform Resource Locator)
- Examples from class
 - Variable name → value
 - Function name → function definition
 - ASCII value → character

Mar 14, 2011

Sprenkle - CSCI111

16

Example: Textbook's Index



Mar 14, 2011

Sprenkle - CSCI111

17

Dictionaries in Python

- Map **keys** to **values**
 - Keys are probably **not** alphabetized
 - Mappings are from **one** key to **one** value
 - Keys are **unique**, Values are not necessarily unique
 - Example: student id → last name
 - Keys must be **immutable** (numbers, strings)
- Similar to Hashtables/Hashmaps in other languages

How would we handle if there is more than one value for a given key?

Mar 14, 2011

Sprenkle - CSCI111

18

Why Dictionaries?

- Another way to store data
- Allow fast lookup of data
 - Requires keys, unique keys
 - Data may not have a natural mapping

Pros	Cons
Fast lookup (<i>much</i> faster than lists if a lot of elements)	Requires a lot of space, unique keys

Mar 14, 2011

Sprenkle - CSCI1111

19

Creating Dictionaries in Python

Syntax:

`{<key>:<value>, ..., <key>:<value>}`

```
empty = {}
ascii = { 'a':97, 'b':98, 'c':99, ..., 'z':122 }
```

Mar 14, 2011

Sprenkle - CSCI1111

20

Dictionary Operations

Indexing	<code><dict>[<key>]</code>
Length (# of keys)	<code>len(<dict>)</code>
Iteration	<code>for <key> in <dict>:</code>
Membership	<code><key> in <dict></code>
Deletion	<code>del <dict>[<key>]</code>

Unlike strings and lists, doesn't make sense to do slicing, concatenation, repetition for dictionaries

Mar 14, 2011

Sprenkle - CSCI1111

21

Dictionary Methods

Method Name	Functionality
<code><dict>.clear()</code>	Remove all items from dictionary
<code><dict>.keys()</code>	Returns a <i>copy</i> of dictionary's list of keys
<code><dict>.values()</code>	Returns a <i>copy</i> of dictionary's list of values
<code><dict>.get(x [, default])</code>	Returns <code><dict>[x]</code> if <code>x</code> is a key; Otherwise, returns <code>None</code> (or default value)

Mar 14, 2011

Sprenkle - CSCI1111

22

Accessing Values using Keys

- Syntax:
`<dictionary>[<key>]`
- Examples:

```
ascii['z']
directory['registrar']
```

- **KeyError** if key is not in dictionary
 - Runtime error; exits program

Mar 14, 2011

Sprenkle - CSCI1111

23

Accessing Values Using `get` Method

- `<dict>.get(x [, default])`
 - Returns `<dict>[x]` if `x` is a key; Otherwise, returns `None` (or default value)

```
ascii.get('z')
directory.get('registrar')
```

- If no mapping, get **None** back instead of **KeyError**

Mar 14, 2011

Sprenkle - CSCI1111

24

Accessing Values

- Typically, you will check if dictionary has a key before trying to access the key

```
if 'Registrar' in phonebook:
    number = phonebook['Registrar']
```

Know mapping exists
before trying to access

- Or handle if get default back

```
number = phonebook.get('Registrar')
if number is None:
    # do something ...
```

No phone number exists

Mar 14, 2011

Sprenkle - CSCI111

25

Recall: Special Value None

- Special value we can use
 - E.g., Return value from function when there is an error
- Similar to **null** in Java

- If you execute

```
list = list.sort()
print list
```

Prints None because `list.sort()` does not return anything

Mar 14, 2011

Sprenkle - CSCI111

26

Example Using None

```
# returns the lowercase letter translated by the key.
# If letter is not a lowercase letter, returns None
def translateLetter( letter, key ):
    if letter < 'a' or letter > 'z':
        return None
    #As usual ...
```

```
# example use
encLetter = translateLetter(char, key)
if encLetter is None:
    print "Error in message: ", char
```

Mar 14, 2011

Sprenkle - CSCI111

27

Inserting Key-Value Pairs

- Syntax:
 - `<dictionary>[<key>] = <value>`
- `ascii['a'] = 97`
 - Creates new mapping of 'a' → 97

ascii_dictionary.py

Mar 14, 2011

Sprenkle - CSCI111

28

Textbook's Index

bookindex["dictionary"] = 58

Keys

	Values
"integer"	20
"list"	60
"string"	45
"float"	25

Mar 14, 2011

Sprenkle - CSCI111

29

Textbook's Index

bookindex["dictionary"] = 58

Keys

	Values
"integer"	20
"list"	60
"string"	45
"dictionary"	58
"float"	25

Mar 14, 2011

Sprenkle - CSCI111

30

Adding/Modifying Key-Value Pairs

- Syntax:
 <dictionary>[<key>] = <value>
- `directory['registrar'] = 8455`
 - Adds mapping for 'registrar' to 8455
- OR
- Modifies old entry if it existed to 8455

Mar 14, 2011

Sprenkle - CSCI111

31

Problem

- Given a file of the form
 - <lastname> <year>
- Create a mapping between the last names and years
 - How do we want to model the data?
 - What is the key? What is the value?
 - How to display the mapping in a pretty way?
 - What order is the data printed in?

`years_dictionary.py`

Mar 14, 2011

Sprenkle - CSCI111

32

Problem

- Modify the previous program to keep track of the *number* of students of each year
 - How do we want to model the data?
 - What is the key? What is the value?
- Could we solve this using a list?

`years_dictionary2.py`

Mar 14, 2011

Sprenkle - CSCI111

33

This Week

- Lab 8
 - List practice
 - Deal or No Deal
- Wednesday, 4 p.m. Talk by Jan Cuny
 - Science Addition G14
 - Answer questions, given Wednesday
 - Next Monday: no class
 - Study for exam!
 - Optional: Meet her at 3 p.m. in Great Hall
- Broader Issue: Sensor Networks

Mar 14, 2011

Sprenkle - CSCI111

34

Analyzing years_dictionary2.py

- Anything useful/general that we could put in a function?

Mar 14, 2011

Sprenkle - CSCI111

35

Problem: Student Majors

- We want to keep track of the number of majors of each type
 - Twist: Not every student has a major (don't declare until sophomore year)

Mar 14, 2011

Sprenkle - CSCI111

`majors_dictionary.py`

36

Problem: Student Majors, revised

- Students can have more than one major
 - Should count these separately
- How can we modify the previous program to do that?

Mar 14, 2011

Sprenkle - CSCI111

37

Thinking about Broader Issue

- Anything useful we could use a dictionary for in the “Cultural Genome” project we read about last week?

Mar 14, 2011

Sprenkle - CSCI111

38

Why Data File Problems Ad Nauseam?

- “**Parsing**” data files for different purposes is very common

Simplified web application access log:

```
128.4.131.54 [09/Aug/2009:14:01:35] GET /dSPACE/simple-search
128.4.133.79 [09/Aug/2009:14:13:13] GET /dSPACE/simple-search
128.4.133.139 [09/Aug/2009:14:28:20] GET /dSPACE/simple-search
128.4.133.139 [09/Aug/2009:14:32:45] GET /dSPACE/adv-search
...
```

I write scripts to

- create user sessions (use as test cases)
- analyze user sessions (avg. length, patterns)
- emulate user sessions

Mar 14, 2011

Sprenkle - CSCI111

39