

Objectives

- Wrap up dictionaries
- Default parameters
- Defining our own classes

Mar 16, 2011

Sprenkle - CSCI111

1

Review: Dictionaries

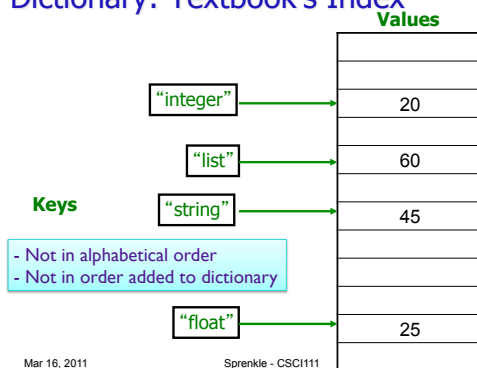
- What is the syntax for creating a new dictionary?
- How do we access a key's value from a dictionary?
 - What happens if there is no mapping for that key?
- How do we create a key → value mapping in a dictionary?
- How do we iterate through a dictionary?

Mar 16, 2011

Sprenkle - CSCI111

2

Dictionary: Textbook's Index



Mar 16, 2011

Sprenkle - CSCI111

3

Review: Creating Dictionaries

Syntax:

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

```
empty = {}
```

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

Mar 16, 2011

Sprenkle - CSCI111

4

Review: Accessing Values Using Keys

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

```
if 'z' in ascii:
    value = ascii['z']
```

Know mapping exists
before trying to access

- Or handle if get default back

```
val = ascii.get('z')
if val is None:
    # do something ...
```

Mar 16, 2011

Sprenkle - CSCI111

5

Equivalent Solutions

```
if key not in dictionary :
    dictionary[key] = 1
else:
    value = dictionary[key] + 1
    dictionary[key] = value
```

```
if key not in dictionary :
    dictionary[key] = 1
else:
    dictionary[key] += 1
```

Mar 16, 2011

Sprenkle - CSCI111

6

Lists vs. Dictionaries

| Lists | Dictionaries |
|---|--|
| integer positions (0, ...) to any type of value | Map immutable keys (int, float, string) to any type of value |
| Ordered | Unordered |
| Slower to find a value (in) | Fast to find a value (use key) |
| Fast to print in order | Slower to print in order (by key) |
| Only as big as you make it | Takes up a lot of space (so can add elements in the middle) |

Mar 16, 2011

Sprenkle - CSCI111

7

PARAMETER DEFAULTS

Mar 16, 2011

Sprenkle - CSCI111

8

Defaults for Parameters

- Can assign a default value to a parameter
 - In general, in function header, default parameter(s) should come *after* all the parameters that *need* to be defined
- Example: **xrange** function
 - Didn't have to specify start or increment when calling the function
 - Default start = 0
 - Default increment = 1

Mar 16, 2011

Sprenkle - CSCI111

9

Using Default Parameters

- By default, the `rollDie` function could assume that a die has 6 sides

Assigns a value to sides
ONLY IF not passed a parameter

```
def rollDie(sides=6):
    return random.randint(1,sides)
```

Examples of calling the function:

```
rollDie(6)
rollDie()
rollDie(12)
```

Show help

Mar 16, 2011

Sprenkle - CSCI111

game.py

10

Getting Documentation

- dir**: function that returns a list of methods and attributes in an object
 - `dir(<type>)`
- help**: get documentation
- In the Python shell
 - `help(<type>)`
 - `import <modulename>`
 - `help(<modulename>)`

Mar 16, 2011

Sprenkle - CSCI111

11

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

Mar 16, 2011

Sprenkle - CSCI111

12

ABSTRACTIONS

Mar 16, 2011

Sprenkle - CSCI111

13

Abstractions

- Provide ways to think about program and its data
 - Get the jist without the details
- Examples we've seen
 - Functions and methods `encodeMessage(phrase, key)`
 - Used to perform some operation but we don't need to know how they're implemented
 - Dictionaries
 - Know they map keys to values
 - Don't need to know how the keys are organized/stored in the computer's memory
 - Just about everything we do in this class...

Mar 16, 2011

Sprenkle - CSCI111

14

Classes and Objects

- Provide an abstraction for how to organize and reason about data
- Example: GraphWin class
 - Had **attributes** (i.e., data or state) background color, width, height, and title
 - Each GraphWin object had these attributes
 - Each GraphWin object had its own values for these attributes
 - Used methods (API) to modify the object's state, get information about attributes

Mar 16, 2011

Sprenkle - CSCI111

15

Defining Our Own Classes

- Often, we want to represent data or information that we do **not** have a way to represent using *built-in types* or *libraries*
- Classes provide way to *organize* and *manipulate* data
 - Organize: data structures used
 - E.g., ints, lists, dictionaries, other objects, etc.
 - Manipulate: methods

Mar 16, 2011

Sprenkle - CSCI111

16

What is a Class?

- Defines a new **data type**
- Defines the class's **attributes** (i.e., data or state) and **methods**
 - Methods are like **functions** *within* a class and are the class's **API**

Internal
data hidden
from others

Object o of
type
Classname

Other objects
manipulate using
methods

Mar 16, 2011

Sprenkle - CSCI111

17

Defining a Card Class

- Create a class that represents a playing card
 - How can we represent a playing card?
 - What information do we need to represent a playing card?



Mar 16, 2011

Sprenkle - CSCI111

18

Representing a Card object

- Every card has two attributes:
 - Suite (one of “hearts”, “diamonds”, “clubs”, “spades”)
 - Rank
 - 2-10: numbered cards
 - 11: Jack
 - 12: Queen
 - 13: King
 - 14: Ace

Mar 16, 2011

Sprenkle - CSCI1111

19

Defining a New Class

- Syntax:

Typically starts with a capital letter

Keyword

```
class ClassName:
    <method definitions>
```

Mar 16, 2011

Sprenkle - CSCI1111

20

Card Class (Incomplete)

```
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 are like functions defined in a class

Mar 16, 2011

Sprenkle - CSCI1111

card.py

21

Defining the Constructor

- `__init__` method is like the **constructor**
 - In constructor, define **instance variables**
 - Data contained in every object
 - Also called **attributes** or **fields**
 - Constructor **never returns** anything
 - First parameter of **every** method is **self**
 - pointer to the object that method acts on
- ```
def __init__(self, rank, suit):
 """Constructor for class Card takes int rank
 and string suit."""
 self.rank = rank
 self.suit = suit
```

Instance variables

Mar 16, 2011

Sprenkle - CSCI1111

22

## Using the Constructor

```
def __init__(self, rank, suit):
```

- As defined, constructor is called using **Card(<rank>, <suit>)**
  - Do not pass anything for the **self** parameter
  - Python handles for us
    - Passes the parameter *automatically*

Object **card**  
of type **Card**

```
rank = ?
suit = ?
```

Mar 16, 2011

Sprenkle - CSCI1111

23

## Using the Constructor

```
def __init__(self, rank, suit):
```

- As defined, constructor is called using **Card(<rank>, <suit>)**
  - Do not pass anything for the **self** parameter
  - Python handles underneath, passing the parameter for us automatically
- Example:
  - **card = Card(2, “hearts”)**
  - Creates a 2 of Hearts card
  - Python passes **card** as **self** for us

Object **card**  
of type **Card**

```
rank = 2
suit = “hearts”
```

Mar 16, 2011

Sprenkle - CSCI1111

24

## Accessor Methods

- Need to be able to get information about the object

- Have **self** parameter
- Return data/information

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

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

- These will get called as **card.getRank()** and **card.getSuit()**
  - Python plugs **card** in for **self**

Mar 16, 2011

Sprengle - CSC1111

25

## Another Special Method: `__str__`

- Returns a **string** that describes the object
- Whenever you **print** an object, Python checks if the object's `__str__` method is defined
  - Prints result of calling `__str__` method
- str(<object>)** also calls `__str__` method

```
def __str__(self):
 """Returns a string
 describing the card as 'rank of
 suit'."""
 result = ""
 if self.rank == 11:
 result += "Jack"
 elif self.rank == 12:
 result += "Queen"
 elif self.rank == 13:
 result += "King"
 elif self.rank == 14:
 result += "Ace"
 else:
 result += str(self.rank)
 result += " of " + self.suit
 return result
```

Mar 16, 2011

Sprengle - CSC1111

26

## Using the Card Class

Invokes the `__str__` method

```
def main():
 c1 = Card(14, "spades")
 print c1
 c2 = Card(13, "hearts")
 print c2
```

Displays:

Ace of spades  
King of hearts

Object c1 of  
type Card

```
rank = 14
suit = "spades"
```

Object c2 of  
type Card

```
rank = 13
suit = "hearts"
```

Mar 16, 2011

Sprengle - CSC1111

27

## Example: Rummy Value

- Problem:** Add a method to the Card class called **rummyValue** that returns the value of the card in the game of Rummy
- Procedure** for defining a method (similar to functions)
  - What is the input?
  - What is the output?
  - What is the method header?
  - What does the method do?
- How do we call the method?

Mar 16, 2011

Sprengle - CSC1111

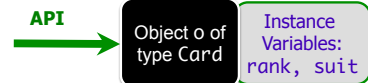
card2.py

28

## Card API

- Based on what we've seen/done so far, what does the Card class's API look like?

## Card API



Implementation of  
methods is hidden

- Card(<rank>, <suit>)**
- getRank()**
- getSuit()**
- rummyValue()**
- \_\_str\_\_()**

Mar 16, 2011

Sprengle - CSC1111

29

Mar 16, 2011

Sprengle - CSC1111

30

## Defining a Card Class

- Create a class that represents a playing card
  - How can we represent a playing card?
  - What information do we need to represent a playing card?
- Do we **need** a class to represent a card?
  - Does any built-in data type naturally represent a card?



Mar 16, 2011

Sprenkle - CSCI111

31

## Using the Card class

- Having the Card class means that we can represent a Card in code

Now that we have the Card class, how can we **use** it?

Mar 16, 2011

Sprenkle - CSCI111

32

## Using the Card class

Now that we have the Card class, how can we **use** it?

- Let's write a simplified version of the game of War
  - Basically just part of a round
- What are the rules of War?

Mar 16, 2011

Sprenkle - CSCI111

war.py

33

## Using the Card class

Now that we have the Card class, how can we **use** it?

- Can make a **Deck** class
  - What data should a Deck contain?
  - How can we represent that data?
- To start: write methods **\_\_init\_\_** and **\_\_str\_\_**
  - What do the method headers look like?

Mar 16, 2011

Sprenkle - CSCI111

34

## This Week

- Jan Cuny's talk – TODAY at 4 p.m., Science Center Addition G14
  - Optional reception at 3 p.m. in Great Hall
  - Answer questions on handout during talk
  - No class on Monday – study for exam
- Lab 8 due Friday
- Broader Issue: environmental monitoring using sensor networks

Mar 16, 2011

Sprenkle - CSCI111

35

## Creating a Deck Class (Partial)

- List of Card objects

```
from card import *

class Deck:
 def __init__(self):
 self.cards = []
 for suit in ["clubs", "hearts", "diamonds", "spades"]:
 for rank in xrange(2,15):
 self.cards.append(Card(rank, suit))

 def __str__(self):
 deckRep = ""
 for c in self.cards:
 deckRep += str(c) + "\n"
 return deckRep
```

Initialize instance variable, self.cards

Creates and returns a string

Displays cards on separate lines

Mar 16, 2011

Sprenkle - CSCI111

36

## Using the Deck Class

- How can we use the Deck that we just wrote?

Mar 16, 2011

Sprenkle - CSCI111

37

## Deck API

- What methods should our Deck class provide?
- What do the method headers look like?
- What should they return?
- How do we implement them?

Mar 16, 2011

Sprenkle - CSCI111

38

## Adding Deck Functionality


- Functionality:
  - Shuffle the cards
  - Deal one card
  - Number of cards remaining
- What do the method headers look like?
- What should they return?
- How do we implement them?

Mar 16, 2011

Sprenkle - CSCI111

39

## Deck API

- Deck()  Constructor
- shuffle()
- draw()
- deal(num\_players, num\_cards)
- numRemaining()
- isEmpty()
- \_\_str\_\_()

Mar 16, 2011

Sprenkle - CSCI111

40

## Algorithm for Creating Classes

1. Identify need for a class
2. Identify state or attributes of a class/an object in that class
  - Write the constructor (`__init__`) and `__str__` methods
3. Identify methods the class should provide
  - How will a user call those methods (parameters, return values)?
    - Develop API
  - Implement methods

Mar 16, 2011

Sprenkle - CSCI111

41