

Objectives

- Wrap up defining classes
- `__lt__`, `__eq__` method
- Helper methods
- Command-line arguments

Mar 21, 2012

Sprengle - CSCI111

1

Review

- Where do we define the data that is needed to represent every object of a class?
 - How do we access that data?
- How do we create a new method?
- What method do you define for the object's constructor?
- What method do you define to be called by print?

Mar 21, 2012

Sprengle - CSCI111

2

Implementing Deck Functionality

- What functionality do we have so far?
- What additional methods should our Deck class provide?
- What do the method headers look like?
 - Deck's API
- What should they return?
- How do we implement them?

Mar 21, 2012

Sprengle - CSCI111

3

`__LT__` and `__EQ__` METHODS

Mar 21, 2012

Sprengle - CSCI111

4

`__eq__`: Compare Objects of Same Type

- Header: **`def __eq__(self, other)`**
 - **Assumption:** `other` is another object of the *same type*
- Returns
 - True if `self` is equivalent to `other`
 - False otherwise
- Similar to implementing `Comparable` interface in Java
- Can now use objects in comparison expressions
 - `==`

Mar 21, 2012

Sprengle - CSCI111

How would you determine if two Card objects are equivalent?

`__lt__`: Compare Objects of Same Type

- Header: **`def __lt__(self, other)`**
 - **Assumption:** `other` is another object of the *same type*
- Returns
 - True if `self < other`
 - False otherwise
- Similar to implementing `Comparable` interface in Java
- Can now use objects in comparison expressions
 - `<`, `sort`

Mar 21, 2012

Sprengle - CSCI111

How do you compare two Card objects?

Comparing Objects of the Same Type

```
def __eq__(self, other):
    """Compares Card objects by their ranks and suits"""
    if type(self) != type(other):
        return False
    return self.rank == other.rank and self.suit == other.suit
# Could compare by black jack or rummy value

def __lt__(self, other):
    """Compares Card objects by their ranks"""
    if type(self) != type(other):
        return False
    return self.rank < other.rank
```

Mar 21, 2012

Sprenkle - CSCI1111 card.py

7

Frequency Object

```
def __lt__(self, other):
    """Compares this object with other, which is
    also a FrequencyObject. Used when using the
    list's sort method."""
    return self.count < other.count
```

Mar 21, 2012

Sprenkle - CSCI1111

8

HELPER METHODS

Mar 21, 2012

Sprenkle - CSCI1111

9

Helper Methods

- Part of the class
- Not part of the API
- Make your code easier but others outside the class shouldn't use
- Convention: method name begins with “_”

Mar 21, 2012

Sprenkle - CSCI1111

10

Example Helper Methods

- Only *loosely* enforces that other can't use
 - Doesn't show up in `help`
 - Does show up in `dir`

Helper Method:

```
def _isFaceCard(self):
    if self.rank > 10 and self.rank < 14:
        return True
    return False

def rummyValue(self):
    if self._isFaceCard():
        return 10
    elif self.rank == 10:
        return 10
    elif self.rank == 14:
        return 15
    else:
        return 5
```

In use:

Mar 21, 2012

card2.py Sprenkle - CS

Summary: Designing Classes

- What does the object/class represent?
- How to model/represent the class's *data*?
 - Instance variable
 - Data type
- What *functionality* should objects of the class have?
 - How will others want to use the class?
 - Put into methods for others to call (API)

Mar 21, 2012

Sprenkle - CSCI1111

12

Discussion

- How did we represent a bug in Lab 6?
- How did we manipulate the bug?
- What was tricky about the implementation?

Mar 21, 2012

Sprenkle - CSCI111

13

Refactoring Bug Class

- What is a bug's data?
- What methods should a Bug object implement?

Mar 21, 2012

Sprenkle - CSCI111

bug.py

14

Benefits of Classes

- Package/group related data into one object
 - Deck can have list of `Card` objects rather than a list of ranks and a list of suits
- Reuse code
 - Card class used in `war.py` and `deck.py`
- Provide interface, can change underlying implementation without affecting calling code

Mar 21, 2012

Sprenkle - CSCI111

15

Changing Implementations

- Same API, different implementations

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

def getRank(self):
    return self.rank

def getSuit(self):
    return self.suit
```

```
def __init__(self, rank, suit):
    self.cardid = rank
    if suit == "clubs":
        self.cardid += 13
    elif suit == "hearts":
        self.cardid += 26
    elif suit == "diamonds":
        self.cardid += 39

def getRank(self):
    return (self.cardid - 2) % 13 + 2

def getSuit(self):
    suits = ["spades", "clubs", "hearts", "diamonds"]
    whichsuit = (self.cardid - 2) / 13
    return suits[whichsuit]
```

Tradeoff: Saving
information (memory);
Computing information

Mar 21, 2012

Sprenkle - CSCI111

card_byid.py

16

Considerations for Using Classes

Only use class if you're using most of its
functionality/information

Since you don't know implementation,
may inadvertently duplicate code

- Redo something done by class
- Could have efficiency penalties
- But time saved reusing code is usually worth it

Mar 21, 2012

Sprenkle - CSCI111

17

COMMAND-LINE ARGUMENTS

Mar 21, 2012

Sprenkle - CSCI111

18

Command-line Arguments

- We can run programs from terminal (i.e., the “command-line”) and from IDLE
- Can pass in arguments from the command-line, similar to how we use Unix commands
 - Ex: `cp <source> <dest>`
 Command-line arguments
 - Ex: `python3 maptest.py 3`
- Makes input easier
 - Don't have to retype each time executed

Mar 21, 2012

Sprenkle - CSCI111

19

Command-line Arguments

- Using the **sys** module
 - What else did we use from the **sys** module?
- ```
python3 maptest.py 3
python3 command_line_args.py <filename>
```
- List of arguments, named **sys.argv**
- How to reference (get value) “<filename>”?

Mar 21, 2012

Sprenkle - CSCI111

20

## Command-line Arguments

- Using the **sys** module
 

```
python command_line_args.py <filename>
```
- sys.argv** →
- |                      |            |
|----------------------|------------|
| command_line_args.py | <filename> |
| 0                    | 1          |
- How to reference (get value) “<filename>”?
    - `sys.argv` is a *list* of the arguments
    - `sys.argv[1]` is the filename
    - `sys.argv[0]` is the name of the program

Mar 21, 2012

Sprenkle - CSCI111 `command_line_args.py`

21

## Using Command-line Arguments

- In general in Python:
    - `sys.argv[0]` is the Python program's name
  - Have to run program from terminal (not from IDLE)
    - Can edit program in IDLE though
- ➔ Useful trick:
- If can't figure out bug in IDLE, try running from command-line
    - May get different error message

Mar 21, 2012

Sprenkle - CSCI111

22

## Questions about Exam 2?

Mar 21, 2012

Sprenkle - CSCI111

23

## Looking Ahead

- Lab 9 Extension
  - Submit everything but graphs, web page by Friday
  - Graphs, web page due Monday
- Exam Friday

Mar 21, 2012

Sprenkle - CSCI111

24