Objectives

- Command-line arguments
- Representing Data
- Broader Issue: Facebook

Command-line Arguments

- Shown that 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 $\underbrace{\text{<source> <dest>}}_{\text{Command-line arguments}}$
  - Ex: python $\overbrace{\text{maptest.py 3}}^{\text{Command-line arguments}}$

Command-line Arguments

- Using the **`sys`** module
  - What else did we use from the **`sys`** module?

python $\underbrace{\text{maptest.py 3}}$

python $\underbrace{\text{command\_line\_args.py <filename>}}_{\textbf{List}\text{ of arguments, named }\texttt{sys.argv}}$

- How to reference (get value) “<filename>”?

Command-line Arguments

- Using the **`sys`** module

python command_line_args.py <filename>

`sys.argv` $\rightarrow$

command_line_args.py	<filename>
0	1

- How to reference (get value) “<filename>”?
  - sys.argv[1]
  - sys.argv[0] is the name of the program

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

Review Code

- Counter class
- Application to the Caesar Cipher

Comparing Objects of the Same Type

- Special **__cmp__** method
  - Header: **`_cmp__(self, other)`**
    - **`other`** is another object of the *same type*
  - Returns
    - Negative integer if self < other
    - 0 if self==other
    - Positive integer if self > other
- Similar to implementing **`Comparable`** interface in Java
- Can now use objects in comparison expressions
  - <,>,==, etc.

Comparing Objects of the Same Type

- Example Code:

```
def __cmp__(self, other):
    """ Compares Card objects by their ranks """
    # Could compare by black jack value or rummy value
   if self.rank < other.getRank():
      return -1
   elif self.rank > other.getRank():
      return 1
   else:
      return 0
```

card3.py

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)

Benefits of Classes

- Package/group related data into one object
  - Can have list of Card objects rather than a list of ranks and a list of suits
- Reusing code
  - E.g., Don't need to check if user put in valid key
- Provide interface, can change underlying implementation without affecting calling code

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
```

Tradeoff: Saving information (memory); Computing information

```
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]
```

card_byid.py

Two More Implementations

- The Counter class
  - Compare counter.py and counter2.py's increment and decrement implementations

Considerations for Using Classes

- Only use class if you're using most of its functionality/information
  - Don't use Counter for validating if a number is within the valid range
    - Because not using the wrapping/current value
- Since 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

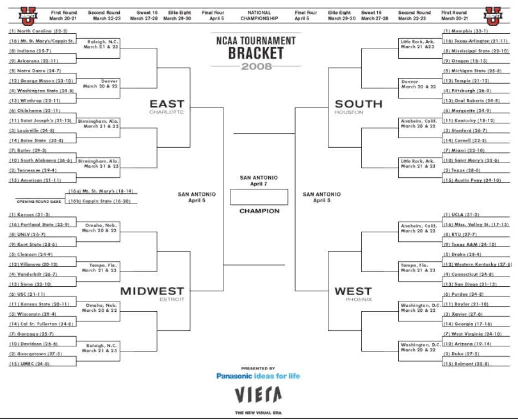
Extra Credit Functionality Ideas

- Return the card's color (Red/Black), using a constant defined at the top for each color
  - What game is this useful for?
- Boolean methods: isBlack(), isRed()
- Boolean method: isOppositeColor(card)
- Boolean method: isSameSuit(card)
- Create a Hand class (very similar to Deck class)
  - Methods that check if all same suit, all same rank
- Player class for various games …
- Test/Demonstrate your methods

Due Tuesday before lab

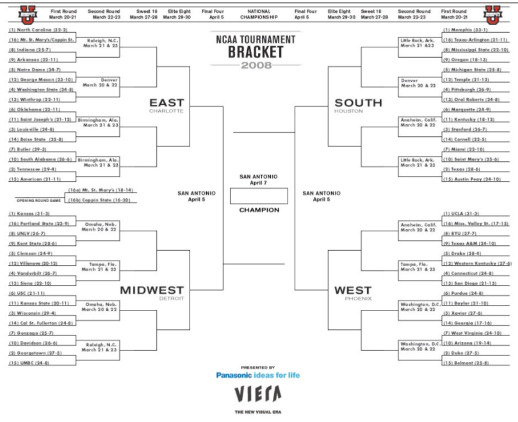
March Madness, Baby!

- NCAA College Basketball Tournament
- 65 teams play for the championship
  - Play in several rounds
  - Duke - champ in 2001
- Broken into 4 brackets
  - 16 teams per bracket
  - Ranked 1-16
  - Games favor higher-ranked teams



Brainstorming on Tournament Info

- What data do we want to represent?
  - What data types would we use to represent this information?
- What do we want to be able to do with this data?

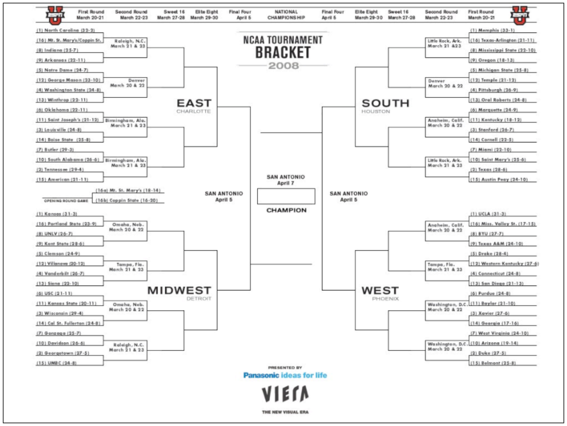


Tournament Info

- Information
  - Teams (name, seed, mascot, record, bracket, …)
  - Brackets (regions)
  - Winners, upsets
  - People’s picks
    - Scores (considering points per round)
- What do you want to do with the data?
  - Whose picks are most accurate?
  - When has someone been mathematically eliminated as the winner?
  - Who did people pick to win overall? Individual game?

Tournament Info

- Tradeoffs in representing information
  - Depends on what you do with it
  - Ease of calculation to answer those questions
- Concise representation of this information
  - Represent people’s picks, check if correct
  - Solution by Patrick Reynolds, maintainer of the *Oracle of Bacon*



Concise Representation

- Represent each game as a boolean (bit)
  - upsets are 1s (0s if “goes to seed”)
  - #1 seeds playing each other: goes by bracket
- 63 games (ignoring play-in)
  - Organize by bracket
  - Break into **bytes** (8 bits)
  - 0 in 64th bit
- For one bracket, first round:

Game	1-16	8-9	5-12	4-13	6-11	3-14	7-10	2-15
Upset?								
Bit Mult	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

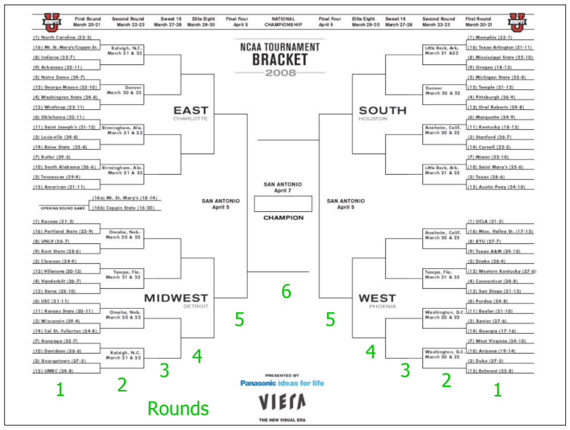
Concise Representation

- For one bracket, in Round 1

Game	1-16	8-9	5-12	4-13	6-11	3-14	7-10	2-15
Upset?	0	1	0	0	0	0	1	0
Bit Mult	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- What each byte represents

Bracket/ Round	East 1	MW 1	South 1	West 1	E, MW 2	S, W 2	All-3	Elite Eight (4), Final Four (2), Final (1)
Byte	1	2	3	4	5	6	7	8



Scenarios

- If a 12-seed beats a 5-seed, that's an upset
- If that 12-seed then beats the 13-seed (who beat a 4-seed), that is also an upset
  - As if 12-seed was the 5th seed beating the 4th seed

Championship

- Representation "favors" UNC over Memphis in the championship
  - Favors East over MidWest and South over West
- To make the code be 0.0.0.0.0.0.0.2
  - All #1s win except Kansas (beaten by Georgetown)
    - Rest plays out as expected

Representation in Python

- How would we represent this information in Python?

Problem:

- How would you determine if someone got a pick right?
- How would you determine how many picks someone got right?
- How would you determine a person's score?

Problem:

- How would you determine if someone got a pick right?
  - Compare bits -- right iff they're the same
- How would you determine how many picks someone got right?
  - Total pairs of bits that are the same
- How would you determine a person's score?
  - Multiply <each pair right> by weight of the round

Advantages of This Representation

- Small (bits are tiny)
- Doesn't change with the teams playing in the tournament
  - Read in a file of team names to represent the brackets (in the required order)
    - East, #1; East, #16; …
  - Separate representation from "configuration"
- Easy to compute correctness and score
- Consistency
  - Can't accidentally have a loser win later

Disadvantages of Representation

- Some queries are more difficult
  - From code, who did someone select as champion?
  - How far did someone predict Duke will go?

Broader Issue

- Facebook's News Feed
- Discussion:
  - What are the pros and cons of the News Feed?
  - What are Facebook's privacy and security issues?
    - How does Facebook address these issues?
  - Why has Facebook been so much more successful than predecessors such as Friendster and Orkut?

Group 1: Greg, Dave, Joe, Colin
Group 2: Alex, Nay, Julie, Vasil
Group 3: Ty, Clay, Arturo
Group 4: Joa, Lucy, Stuart

Discussion

- Good algorithm → Business success
  - Google's PageRank algorithm
    - Revenue: $16 billion (2007)
  - Facebook's Newsfeed algorithm
    - Revenue: $150 million/year
  - Others?

Discussion

- Algorithm uses
  - Lots of data --> how is it organized?
  - Fancy frequency tables
    - we have used simplified versions
  - Weight factors (Deal or No Deal offer)
  - AI to adapt the weights
- Frequency algorithms, most-recent algorithms: commonly used in OS, Architecture (caching)
- Be careful with Facebook (and MySpace and others) when you're job hunting