Markov Text Generation (cont.); Board Objects for Connect Four

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

ps8pr3: *Generate* Text Based on Shakespeare ...Or Anyone Else!

```
Boston University is an international, comprehensive, private research
university, committed to educating students to be reflective, resourceful
individuals ready to live, adapt, and lead in an interconnected world. Boston
University is committed to generating new knowledge to benefit society.

We remain dedicated to our founding principles: that higher education should be
accessible to all and that research, scholarship, artistic creation, and
professional practice should be conducted in the service of the wider community—
local and international. These principles endure in the University's insistence
on the value of diversity, in its tradition and standards of excellence, and in
its dynamic engagement with the City of Boston and the world.

Boston University comprises a remarkable range of undergraduate, graduate, and
professional programs built on a strong foundation of the liberal arts and
sciences. With the support and oversight of the Board of Trustees, the
University, through our faculty, continually innovates in education and research
to ensure that we meet the needs of students and an ever-changing world.
```

mission.txt

```
>>> d2 = create_dictionary('mission.txt')

>>> generate_text(d2, 20)
We remain dedicated to benefit society. Boston
University is an ever-changing world. Boston University
comprises a strong foundation of diversity,
```

…which of these could be one of the entries in d?

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```
edited_mission.txt

```
>>> d = create_dictionary('edited_mission.txt')
```

A. `'a': ['comprehensive']`

B. `'It': ['is']`

C. `'knowledge.': ['new']`

D. two of the above (which ones?)

E. A, B, and C

Which of these could be one of the entries in d?

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```
edited_mission.txt

```
>>> d = create_dictionary('edited_mission.txt')
```

key = a word w

value = a list of the words that follow w in the text

A. **`'a': ['comprehensive']`** yes

B. ~~`'It': ['is']`~~ `'It': ['is', 'is', 'is']`

C. ~~`'knowledge.': ['new']`~~ `'new': ['knowledge.']`

D. two of the above (which ones?)

E. A, B, and C

What about this one?
`'knowledge.': ['It']`
no! sentence-ending words are not used as keys.

A Markov Model in Dictionary Form

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```

edited_mission.txt

sentence-start symbol

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating', 'be', 'live', 'lead', 'generating'],
 'committed': ['to', 'to'],
 ... }
```

key = a word w

value = a list of the words that follow w in the text

- *Sentence-ending words* should not be used as keys.
  - words that end with a `'.'`, `'?'`, or `'!'` (e.g., `'world.'`)

Model Creation Function

```
def create_dictionary(filename):
    # read in file and split it into a list of words

    d = {}
    current_word = '$'

    for next_word in words:
        if current_word not in d:
            d[current_word] = [next_word]
        else:
            d[current_word] += [next_word]

        # update current_word to be either
        # next_word or '$'...

    return d
```

key = a word w

value = a list of the words that follow w in the text

Model Creation Example

```
words = ['Boston', 'University', 'is', 'a', 'comprehensive',
   'university.', 'It', 'is', 'committed', ...]
d = {}
current_word = '$'

for next_word in words:
    if current_word not in d:
        d[current_word] = [next_word]
    else:
        d[current_word] += [next_word]

    # update current_word to be either next_word or '$'...
```

current_word	next_word	action taken
'$'	'Boston'	d['$'] = ['Boston']
'Boston'	'University'	d['Boston'] = ['University']
'University'	'is'	d['University'] = ['is']
'is'	'a'	d['is'] = ['a']
'a'	'comprehensive'	d['a'] = ['comprehensive']

Model Creation Example

```
words = ['Boston', 'University', 'is', 'a', 'comprehensive',
   'university.', 'It', 'is', 'committed', ...]
d = {}
current_word = '$'

for next_word in words:
    if current_word not in d:
        d[current_word] = [next_word]
    else:
        d[current_word] += [next_word]

    # update current_word to be either next_word or '$'...
```

current_word	next_word	action taken
'$'	'Boston'	d['$'] = ['Boston']
'Boston'	'University'	d['Boston'] = ['University']
'University'	'is'	d['University'] = ['is']
'is'	'a'	d['is'] = ['a']
'a'	'comprehensive'	d['a'] = ['comprehensive']
'comprehensive'	'university.'	d['comprehensive']=['university.']

Model Creation Example

```
words = ['Boston', 'University', 'is', 'a', 'comprehensive',
   'university.', 'It', 'is', 'committed', ...]
d = {}
current_word = '$'

for next_word in words:
    if current_word not in d:
        d[current_word] = [next_word]
    else:
        d[current_word] += [next_word]

    # update current_word to be either next_word or '$'...
```

current_word	next_word	action taken
'$'	'Boston'	d['$'] = ['Boston']
'Boston'	'University'	d['Boston'] = ['University']
'University'	'is'	d['University'] = ['is']
'is'	'a'	d['is'] = ['a']
'a'	'comprehensive'	d['a'] = ['comprehensive']
'comprehensive'	'university.'	d['comprehensive']=['university.']
'$'	'It'	d['$'] → ['Boston', 'It']

Model Creation Example

```
words = ['Boston', 'University', 'is', 'a', 'comprehensive',
   'university.', 'It', 'is', 'committed', ...]
d = {}
current_word = '$'

for next_word in words:
    if current_word not in d:
        d[current_word] = [next_word]
    else:
        d[current_word] += [next_word]

    # update current_word to be either next_word or '$'...
```

current_word	next_word	action taken
'$'	'Boston'	d['$'] = ['Boston']
'Boston'	'University'	d['Boston'] = ['University']
'University'	'is'	d['University'] = ['is']
'is'	'a'	d['is'] = ['a']
'a'	'comprehensive'	d['a'] = ['comprehensive']
'comprehensive'	'university.'	d['comprehensive']=['university.']
'$'	'It'	d['$'] → ['Boston', 'It']
'It'	'is'	d['It'] = ['is']
'is'	'committed'	d['is'] → ['a', 'committed']

Using the Model to Generate New Text

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```

edited_mission.txt

- Here's a portion of our Markov model for the above text:

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating','be','live','lead','generating'],
 'committed': ['to', 'to'],
 'It': ['is', 'is', 'is'], … }
```

- We use it to generate new text:

Using the Model to Generate New Text

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```

edited_mission.txt

- Here's a portion of our Markov model for the above text:

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating','be','live','lead','generating'],
 'committed': ['to', 'to'],
 'It': ['is', 'is', 'is'], … }
```

- We use it to generate new text:
  `It`

Using the Model to Generate New Text

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```
edited_mission.txt

- Here's a portion of our Markov model for the above text:

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating','be','live','lead','generating'],
 'committed': ['to', 'to'],
 'It': ['is', 'is', 'is'], … }
```

- We use it to generate new text:

```
It is
```

Using the Model to Generate New Text

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```
edited_mission.txt

- Here's a portion of our Markov model for the above text:

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating','be','live','lead','generating'],
 'committed': ['to', 'to'],
 'It': ['is', 'is', 'is'], … }
```

- We use it to generate new text:

```
It is amazing!
```

Using the Model to Generate New Text

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```

edited_mission.txt

- Here's a portion of our Markov model for the above text:

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating','be','live','lead','generating'],
 'committed': ['to', 'to'],
 'It': ['is', 'is', 'is'], … }
```

- We use it to generate new text:

```
It is amazing!
```

Using the Model to Generate New Text

```
Boston University is a comprehensive university.
It is committed to educating students to be ready
to live and to lead in an interconnected world.
It is committed to generating new knowledge.
It is amazing!
```

edited_mission.txt

- Here's a portion of our Markov model for the above text:

```
{'$': ['Boston', 'It', 'It', 'It'],
 'Boston': ['University'],
 'University': ['is'],
 'is': ['a', 'committed', 'committed', 'amazing!'],
 'to': ['educating','be','live','lead','generating'],
 'committed': ['to', 'to'],
 'It': ['is', 'is', 'is'], … }
```

- We use it to generate new text:

```
It is amazing! Boston…
```

generate_text(word_dict, num_words)

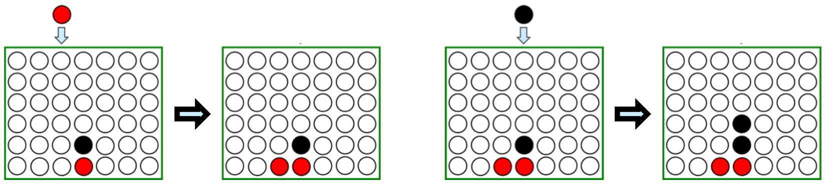
```
start with current_word = '$'

repeat num_words times:
    wordlist = words that can follow current_word
               (use the word_dict dictionary!)
    next_word = random.choice(wordlist)
    print next_word, followed by a space   (use end=' ')
    update current_word to be either next_word or '$'

print()
```

More Generated Shakespeare!

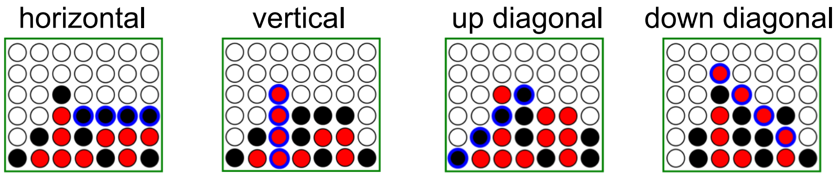
```
LADY CAPULET: Young Romeo is Romeo, he's a good lady,
such sight to stand: and be married?

I hate hell, all in that follows there, and well-
govern'd youth:

BENVOLIO: I'll be rough with me. Rest you without eyes,
Nor bide th'encounter of untimely death:
```

PS 9: Connect Four!

- Two players, each with one type of checker
- 6 x 7 board that stands vertically
- Players take turns dropping a checker into one of the board's columns.



- Win == four adjacent checkers in any direction:



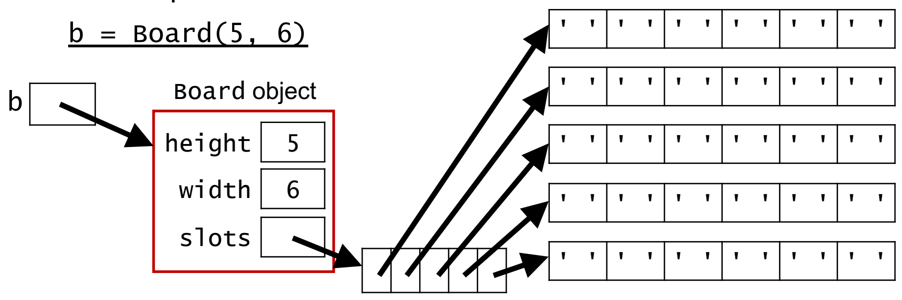
Recall: Classes and Objects

- A *class* is a blueprint – a definition of a new data type.
- We can use the class to create one or more *objects.*
  - "values" / *instances* of that type
- One thing we'll need: a `Board` class!

Board Objects

- To facilitate testing, we'll allow for dimensions other than 6 x 7.
  - example:

```
b = Board(5, 6)
```

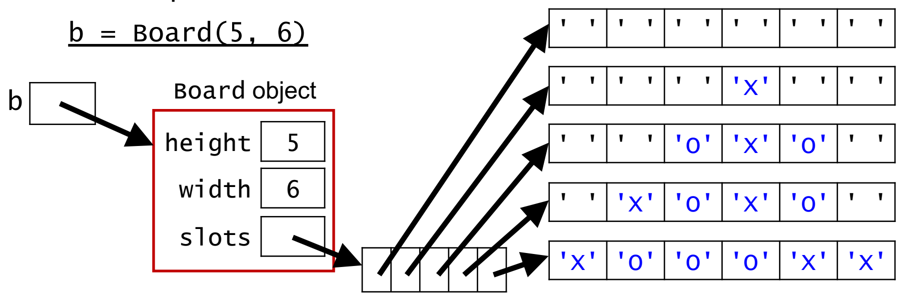


- `slots` is ***a 2-D list*** of single-character strings!
  - `' '` (space) for empty slot
  - `'X'` for one player's checkers
  - `'O'` (not zero!) for the other's

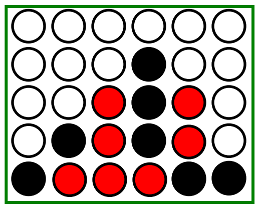
Board Objects

- To facilitate testing, we'll allow for dimensions other than 6 x 7.
  - example:

```
b = Board(5, 6)
```

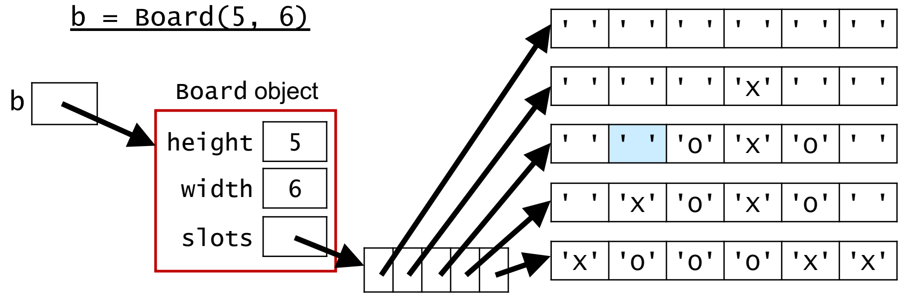


- `slots` is ***a 2-D list*** of single-character strings!
  - `' '` (space) for empty slot
  - `'X'` for one player's checkers
  - `'O'` (not zero!) for the other's



From a Client, How Could We Set the Blue Slot to `'X'`?

`b = Board(5, 6)`



A. `Board.slots[3][2] = 'X'`

B. `Board.slots[1][2] = 'X'`

C. `slots[2][1] = 'X'`

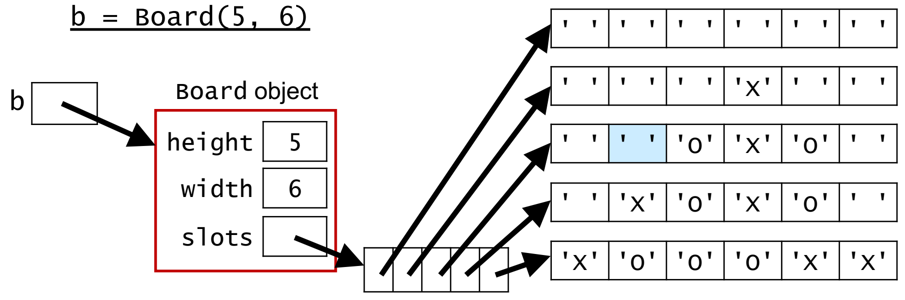
D. `b.slots[1][2] = 'X'`

E. `b.slots[2][1] = 'X'`

How would you write this assignment if it were *inside* a `Board` method?

From a Client, How Could We Set the Blue Slot to `'X'`?

`b = Board(5, 6)`



A. `Board.slots[3][2] = 'X'`

B. `Board.slots[1][2] = 'X'`

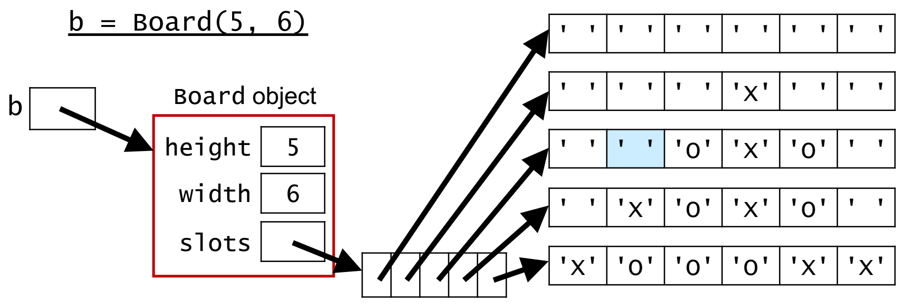
C. `slots[2][1] = 'X'`

D. `b.slots[1][2] = 'X'`

E. `b.slots[2][1] = 'X'`

How would you write this assignment if it were *inside* a `Board` method?

From a Client, How Could We Set the Blue Slot to `'X'`?



A. `Board.slots[3][2] = 'X'`

B. `Board.slots[1][2] = 'X'`

C. `slots[2][1] = 'X'`

D. `b.slots[1][2] = 'X'`

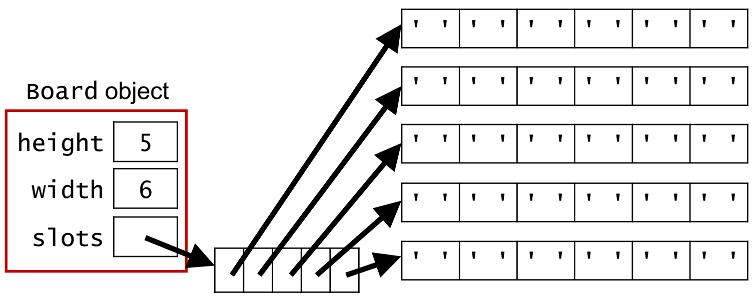
E. `b.slots[2][1] = 'X'`

How would you write this assignment if it were *inside* a `Board` method?

`self.slots[2][1] = 'X'`

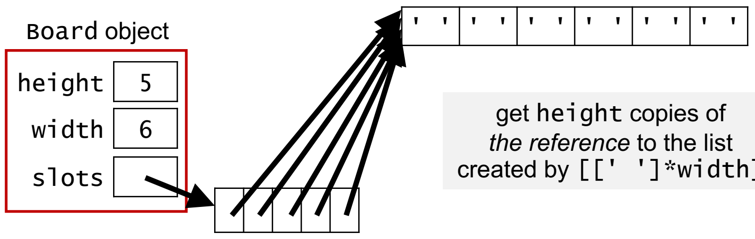
Board Constructor

```
class Board:
    """ a data type for a Connect Four board with
        arbitrary dimensions
    """
    def __init__(self, height, width):
        """ a constructor for Board objects """
        self.height = height
        self.width = width
        self.slots = [[' ']*width] * height  # okay?
```



Board Constructor

```
class Board:
    """ a data type for a Connect Four board with
        arbitrary dimensions
    """
    def __init__(self, height, width):
        """ a constructor for Board objects """
        self.height = height
        self.width = width
        self.slots = [[' ']*width] * height  # okay? no!
```

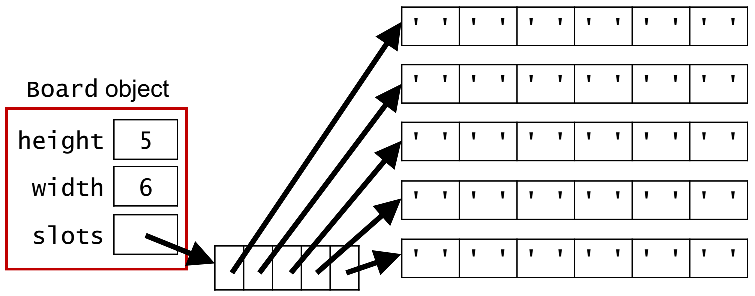


get height copies of *the reference* to the list created by `[[' ']*width]`

Board Constructor

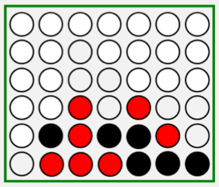
```
class Board:
    """ a data type for a Connect Four board with
        arbitrary dimensions
    """
    def __init__(self, height, width):
        """ a constructor for Board objects """
        self.height = height
        self.width = width
        self.slots = [[' ']*width for r in range(height)]
```

a list comprehension!



__repr__ Method

```
def __repr__(self):
    """ returns a string representation of a Board """
    s = ''    # begin with an empty string

    for row in range(self.height):
        s += '|'
        for col in range(self.width):
            s += self.slots[row][col] + '|'
        s += '\n'

    # add the row of hyphens to s
    # add the column indices to s

    return s
```

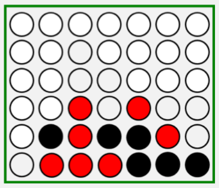


```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |o| |o| | |
| |x|o|x|x|o| |
| |o|o|o|x|x|x|
```

__repr__ Method

```
def __repr__(self):
    """ returns a string representation of a Board """
    s = ''    # begin with an empty string

    for row in range(self.height):
        s += '|'
        for col in range(self.width):
            s += self.slots[row][col] + '|'
        s += '\n'

    # add the row of hyphens to s
    # add the column indices to s

    return s
```

← you'll add code here!



```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |o| |o| | |
| |x|o|x|x|o| |
| |o|o|o|x|x|x|
---------------
 0 1 2 3 4 5 6
```