On your way in...

Pick-up
1. Lecture 15 notes
Welcome to CS 134!

Introduction to Computer Science
Iris Howley

-Classes for generating human-like text-

Spring 2019
HW5 DUE WEDNESDAY
ENCAPSULATING DATA IN CLASSES TO GENERATE HUMAN-LIKE TEXT

Classes + Dictionaries + Generators + Files
Using multiple instances of a text-generating class to generate text for different characters in the same book.
Hagrid: "member. I should n'ta told yeh that! He blurted out. I want yer ticket fer hogwarts he said."
Harry: "n't talk. I tried to turn him yellow yesterday to make him id — he 's runnin' back up ter the school."
Vernon: "aid uncle vernon " so all aboard! " where 's the cannon? you are boy. platform nine—platform ten."
Quirrell: "ee the stone...i 'm presenting it to my master...but where is vering treble either but cold and sharp."
Dumbledore: "dungeons between you and professor quirrell is a complete secret prises even me sometimes...now enough questions."
williams

wi → w → i
il → i → l
ll → l → l → li
li → l → | → l → li
ia → i → a
am → a → m
ms → m → s
s_ → s → _
What does this represent?

The distribution, from our data, of letters that can follow a given letter sequence

We can use this to randomly generate similar text as the original.

A fingerprint
1. Randomly select a letter
2. Let’s say we select ‘i’
3. Given i, the next letter can be
   1. either ‘l’ or ‘a’ with 50% chance each
4. Let’s say we randomly pick ‘a’
5. Now, given ‘a’, what can we choose?
   1. Only m!
6. And then s follows
7. s doesn’t have an entry, so what do we do?
   - We can randomly pick a new letter
   - ...start again from the top, until we decide to stop!

Given more data, our output (blue) will have more possible outcomes.
N-grams

- We’ve been working with 2-grams or bigrams
- There’s also trigrams:
  - wil, ill, lli, lia, iam, ams
- Which we can turn into a distribution as follows
  - wi → l, il → l, ll → i, li → a, ia → m, am → s
- 4-grams, 20-grams, etc. etc.
- We call these “n-grams”
What will we need to build our text-generating **ORACLE** ??

- Choose an n for our n-gram
- Some text to build the letter distribution → file input!
- A data structure to hold the letter distribution/fingerprint
- Somewhere to start generating new text
- Something to do when we run out of letters (i.e. what comes after the ‘s’ in Williams?)
  - We’ll need to store our text
Look at oracle.py

• __slots__ = [ "_corpus", "_dist", "_n" ]
  ▪ A special list to hold the class’ attributes
  ▪ It restricts the attributes to just these!

• __XXXX__ are special python variables/functions
  ▪ __name__, __all__, __slots__, many others!
  ▪ _XXX are variables/functions we don’t want to be public
  ▪ Won’t show up in pydoc3, etc. Just for our use in this Oracle class!
```python
__slots__ = []

>>> class Yesteryears:
...     """ demo of classes from last week """
... 
>>> yy = Yesteryears()
>>> yy.start = 2018
>>> yy.end = 2022
>>> yy.mid = 2020
>>> yy.whatev = "I do what I want!"

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Years' object has no attribute 'mid'
```

Why do these differ?
When we create an instance of a class, that class’s `__init__()` method is called.

```python
g = Oracle()
g.scan(text)
for line in g.lines():
    print(line)
```
def __init__(self):
self is always passed to class methods

• self refers to this particular object (i.e., an object reference)
  """Initialize the oracle with n-gram size n.""
  self._n = n

• When we see self.something, we know it’s a variable, method, etc.
  associated with a particular instance of a class
def __init__(self):

def __init__(self, n = 4):
    """Initialize the oracle with n-gram size n."""
    self._n = n
    self._dist = dict()
    self._corpus = ""

__init__ sets the initial values for attributes within the instance of the class
def __init__(self):

def __init__(self, n = 4):
    """Initialize the oracle with n-gram size n."""
    self._n = n
    self._dist = dict()
    self._corpus = ""

Can be used to set default values in case the user doesn’t pass an argument
How we interact with Oracle

```python
g = Oracle()
g.scan(text)
for line in g.lines():
    print(line)
```

What does this line imply about Oracle’s `lines()` method?
```python
for line in myoracle.lines():

• `.lines()` is a generator
  ▪ yields lines instead of returning lines!
• Our `.lines()` function will produce a generated line of text that will fit on a single line on the console (70-80 characters)
• But we still need a way to generate individual letters to put in the line
  ▪ `for <variable> in <sequence>`
  ▪ `iter()` function is called on the `<sequence>` object
    o Review Lecture 13 on Iterators
    o `iter()` → python actually looks for the object’s `.__iter__()` method
  ▪ It also yields an element from the sequence, one at a time
```
Why use classes instead of a pile of functions?

• Encapsulation! Abstraction!
  • \texttt{g = Oracle()}
  • \texttt{g.scan(text)}
  • \texttt{for line in g.lines():}
    • \texttt{print(line)}

• Maintaining state
  • But we must write our methods to maintain that state

• Lots of fun stuff in software design
Why use classes instead of a pile of functions?

- Multiple oracles at the same time!

```python
ha = Oracle()
...  
print(next(ha.lines()))

he = Oracle()
...  
print(next(he.lines()))
```

```python
Draco Oracle
```

```python
dr = Oracle()
...  
print(next(dr.lines()))
```

```python
Hermione Oracle
```
Classes

>>> Class Years:
   """ Define some attributes """

>>> y = Years()

>>> y
   <__main__.Years object at 0x108c63860>

>>> from oracle import Oracle

>>> o = Oracle()

   <oracle.Oracle object at 0x103485e48>

This is __name__!!!!
On your way out...

Pick-up
1. Graded Midterm Exams
The boxplot shows the distribution of exam scores. A quarter of the class scored above 89%, another quarter scored between 81-89%, and the median score is 81%. The mean score is 80%. The box plot is centered around the median score, indicating a relatively normal distribution of scores.
QUESTIONS?
Leftover Slides
Classes

```python
>>> from oracle import Oracle
>>> o = Oracle()
• >>> type(o)
• <class 'oracle.Oracle'>
```

• `o` is an instance of the class, `Oracle`
• Classes are user-defined types
Classes

```python
>>> from oracle import Oracle
>>> o = Oracle()
>>> o
<oracle.Oracle object at 0x103485e48>
```

...Define the `__repr__()` function in the `oracle` class

```python
>>> from oracle import Oracle
>>> o = Oracle()
>>> o
REPR(): Oracle(n=4)
```
Selecting an item from a sequence

```python
>>> from random import choice
>>> l = ['a', 'b', 'c', 'd']
>>> print(choice(l))
b
>>> print(choice(l))
d
>>> print(choice(l))
a
>>> print(choice(l))
c
>>> print(choice(l))
b
>>> s = "The mountains!"
>>> print(choice(s))
T
>>> print(choice(s))
a
>>> print(choice(s))
n
>>> print(choice(s))
a
>>> print(choice(s))!
```
Shannon Entropy

- Average rate at which information is produced by our data
  - The unexpectedness of a sequence of characters we select

- The **entropy** of a random variable is calculated with this formula:
  1. Where $p_i$ is the probability of seeing a given n-gram in our data
  2. Given a set of n observations
    - Where each observation is a different sequence of characters observed in our data
  3. Compute $p_i$ for the range all observations multiply by $\log_2(p_i)$
  4. Sum across all values