On your way in...

Pick-up:
1. Homework 6
2. Mid-semester feedback form
3. Lecture 16 notes

Drop-off:
1. Completed Homework 5
   • (2 piles, IDs < 50 and IDs >= 50
Welcome to CS 134!

Introduction to Computer Science
Iris Howley

-Methods for making class data more accessible-

Spring 2019
HW6 DUE WEDNESDAY
(AFTER SPRING BREAK)
Midterm Exam Scores

• Issues limited to anonymous IDs 50-60.
• The score on the back sheet of your exam should be no more than a few points lower than the percent written on the front page
  • ex: If there’s an 85 marked on your last page, an 89 is incorrect
  • ex2: If there’s an 85 on your last page, an 83 is incorrect

• No one’s score was curved down
• The curve was limited to only a few points

Contact me if you’d like to confirm your grade
THINKING LIKE A COMPUTER SCIENTIST

return vs. print vs.
yield vs. None
What do you notice?

```python
def printGameboard(board):
    """A method that accepts a list of lists and prints the entire gameboard according to specification."""
    !!! Explicitly says “print”
    We don’t use print(..) that often...
    Except in lecture, to debug, etc.
```
What do you notice?

• `>>> def printFunction(s):
  ...    print(s)
• `>>> test = printFunction("heyyyy")
• `heyyyy
• `>>> test

• `What’s stored in test?

• `>>> print(test)
• `None

• `What will this print?

Even if we don’t say “return None” in our `printFunction(s) definition, it returns None!
We’ve seen this before!

- \( l = [1,2,3] \)

- \( l = l.append(2019) \)

- \( \text{type}(l) \quad \text{If we print}(l) \text{ it will print ‘None’} \)
  - <class 'NoneType'>

Functions always return something.
If you don’t type return or yield, it will return None.
What do you notice?

```python
def parseDegree(d):
    """Given a comma-separated degree, converts it to a 3-element list.

    Returns an empty list ([[]) if degree is empty.

You should contribute additional unit tests below:
>>> len(parseDegree('1988, Ph.D., University of Massachusetts, Amherst'))
3

!!! If parseDegree used print(...) instead of return, len(parseDegree(...)) would not be 3!
```
def randomChar(self):
    """ Draw a random character from the corpus. """

    6. Write a private helper method, _randomChar(self) that returns a char

def _next(self, key):
    """Compute a character that follows the n-1 character key."""
Return is not always so explicit

```python
def scan(self, text):
    """Scan the text and compute the distribution of n-grams.""
```

Here, "compute" does not imply returning something!

5. We will write, together, the method, `scan(self, s)`, that takes a string, `s`, and records the occurrences of `n` character combinations, where `n` is the n-gram size of the Oracle, `self`. Think about how the scanning of `s` should update the other slots of the object. It's important that every method leaves the Oracle in a consistent state.
What do you notice?

```python
def lines(self, min=70, max=80):
    """Generate lines according to the fingerprint. Lines are 70–80 characters long and ideally end on a word boundary."""
```

Write a method, `lines(self, min=70, max=80)`. This is also a generator, but it yields whole lines—strings of between `min` and `max` characters. (The defaults for `min` and `max` should be 70 and 80.) After
Functions vs. Methods in Python

• Methods
  • Functions that are attached to an object
  • Accessed via dot notation (i.e., list.sort())
  • Defined within a class

• Functions
  • A sequence of statements that execute in a certain order, given a name
  • Helpful for code reusability, abstraction, etc.
DIFFERENT KINDS OF METHODS FOR MANAGING DIFFERENT KINDS OF OBJECTS’ DATA

Annotations:
Properties + Setters
Properties

class Pt(object):
    __slots__ = ['_x','_y']
    def __init__(self, x, y):
        self._x = x
        self._y = y

alice = Pt(3,4)

Naming conventions
• Variables start with a lower case
• Classes start with upper case
Properties

class Pt(object):
    __slots__ = ['_x','_y']
    def __init__(self, x, y):
        self._x = x
        self._y = y

alice = Pt(3,4)
alice._x
Accessing Alice’s private attributes!
Properties

class Pt(object):
__slots__ = ['_x','_y']
def __init__(self, x, y):
    self._x = x
    self._y = y

@property
def x(self):
    print("in def x!")
    return self._x

alice = Pt(3,4)
alice.x

Will now go through
the @property and print “in def x”
(removethis before moving on)
Properties

class Pt(object):
    __slots__ = ['_x','_y']
def __init__(self, x, y):
    self._x = x
    self._y = y

alice = Pt(3, 4)
alice.x = 20

AttributeError: can't set attribute

Can read it, but can’t write to it!
Properties

class Pt(object):
    __slots__ = ['_x','_y']
    def __init__(self, x, y):
        self._x = x
        self._y = y

@x.setter  Repeat for Pt._y!
def x(self, v):
    self._x = v

alice = Pt(3,4)
alice.x = 20

Now we can get the private variables via a public property

And we can set the private variables via public setter methods
Properties

class Pt(object):
    __slots__ = ['_x','_y']
def __init__(self, x, y):
    self._x = x
    self._y = y

print("Pt({},{})".
      format(alice.x,alice.y))

This gets old fast!
What if I want to print several points?
class Pt(object):
    __slots__ = ['_x', '_y']
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def __str__(self):
        return "Pt({}, {})".format(self.x, self.y)
Special Methods

class Pt(object):
    __slots__ = ['_x','_y']
def __init__(self, x, y):
    self._x = x
    self._y = y

alice = Pt(3,4)
bob = Pt(3,4)
alice is bob ➔ False
alice == bob ➔ What will this print?
    ➔ False!!

== defaults to ‘is’ when the special method isn’t defined!
Special Methods

class Pt(object):
    __slots__ = ['_x','_y']
    def __init__(self, x, y):
        self._x = x
        self._y = y
    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

alice = Pt(3,4)
bob = Pt(3,4)
alice == bob  # True
== special method is __eq__(..)
Special Methods

- `str(myObject)`
  - `myObject.__str__(self)`
    - Is actually calling

- `repr(myObject)`
  - `myObject.__repr__(self)`

- `len(myObject)`
  - `myObject.__len__(self)`

- `for _ in myObject`
  - `iter(myObject)`
    - `myObject.__iter__(self)`

- `object1 == object2`
  - `object1.__eq__(self, object2)`

- `object1 < object2`
  - `object1.__lt__(self, object2)`

- `object1 >= object2`
  - `object1.__ge__(self, object2)`

- `object1 != object2`
  - `object1.__ne__(self, object2)`
Special Methods

class Pt(object):
    __slots__ = ['_x','_y']
    def __init__(self, x, y):
        self._x = x
        self._y = y

def __add__(self, other):
    return Pt(self.x + other.x, self.y + other.y)

alice = Pt(3,4)
bob = Pt(3,4)
alice + bob

Similar problem!
+ special method is __add__(..)
Overloading Operators

- `obj1 + obj2`
  - `obj1.__add__(self, obj2)`
  - *Is actually calling*
- `obj1 – obj2`
  - `obj1.__sub__(self, obj2)`
- `obj1 * obj2`
  - `obj1.__mul__(self, obj2)`
- `obj1 % obj2`
  - `obj1.__mod__(self, obj2)`
- `object1 and object2`
  - `object1.__and__(self, object2)`
- `object1 or object2`
  - `object1.__or__(self, object2)`
- Etc. etc.

https://docs.python.org/2/reference/datamodel.html#emulating-numeric-types
Tuples, Strings, other built-in types aren’t particularly special!

You can build your own!
On your way out...

PLEASE DROP OFF THE MID-SEMESTER FEEDBACK FORM

(Thanks for your insights!)

PICK UP: Graded Homework 4 (front table)
QUESTIONS?
Leftover Slides