On your way in...

Pick-up:
1. Lecture 17 notes
HW6 DUE WEDNESDAY
(AFTER SPRING BREAK)
Welcome to CS 134!

Introduction to Computer Science

Iris Howley

-Making our own data structures-

Spring 2019
DIFFERENT KINDS OF METHODS FOR MANAGING DIFFERENT KINDS OF OBJECTS' DATA

Finishing up from Wednesday
Properties

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

@property Repeat for Pt._y!
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”
(remove the print before moving on)
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

Python knows which version of def x to call based on usage!
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  # 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)`
- `myList[0]`
  - `myList._getitem__`(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__(..)
More Special Methods

- \( \text{obj1 + obj2} \)  
  - \( \text{obj1.__add__(self, obj2)} \)
  
  *is actually calling*

- \( \text{obj1 – obj2} \)  
  - \( \text{obj1.__sub__(self, obj2)} \)

- \( \text{obj1 * obj2} \)  
  - \( \text{obj1.__mul__(self, obj2)} \)

- \( \text{obj1 % obj2} \)  
  - \( \text{obj1.__mod__(self, obj2)} \)

- \( \text{object1 and object2} \)  
  - \( \text{object1.__and__(self, object2)} \)

- \( \text{object1 or object2} \)  
  - \( \text{object1.__or__(self, object2)} \)

- 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!
What is a list?

class **list**(object)
    list() -> new empty list
    list(Iterable) -> new list initialized from iterable's items

Methods defined here:

    **__add__**(self, value, /)
        Return self+value.

    **__contains__**(self, key, /)
        Return key in self.

    **__delitem__**(self, key, /)
        Delete self[key].

    **__eq__**(self, value, /)
What is a list?
What is a list?

What is the last elephant holding onto?

None
What is a list?

class Element:

_\texttt{value}

\begin{tabular}{c}
3 \\
\_next
\end{tabular}

_\texttt{value}

\begin{tabular}{c}
15 \\
\_next
\end{tabular}

_\texttt{value}

\begin{tabular}{c}
2019 \\
\_next
\end{tabular}
Linked Lists

- See example code in shared/examples/03.15!
class Element:

_class_value: 3
_class_next:

_class_value: 15
_class_next:

_class_value: 2019
_class_next:

_class_head:
Tuples, Strings, other built-in types aren’t particularly special!

You can build your own!
Thought question:
How would you build a doubly-linked list?
QUESTIONS?
For..loops

• for item in mylist:
  ▪ print(item)

This is really:

• try:
  ▪ it=iter(mylist)
  ▪ while True:
    o item = next(it)
    o print(item)

• except StopIteration:
  ▪ pass
For..loops

- for item in mylist:
  - print(item)

- try:
  - it=mylist.__iter__()
  - while True:
    - item = next(it)
    - print(item)
  - except StopIteration:
    - pass

- for ...in myList: Gets an iterator for the list by calling myList.__iter__()
- At the top of the loop, item is updated with the next() item from the generator
- The loop repeats until a StopIteration exception is encountered
  - If the list were endless, the for..loop would never stop!
Steps for Recursion

• Know when to stop.
• Decide how to take one step.
• Break the journey down into that step plus a smaller journey.
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”
(remove this 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

Python knows which version of def x to call based on usage!
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?
Special Methods

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)

print(alice)
Pt(3, 4)
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!