Object-oriented Programming and Python Classes

Lecturer: Chris Terman

Objects!

Python supports many different kinds of data:

```
1234 int          3.14159 float          "Hi there!" str

[1, 2, 3, 5, 7, 11] list

{"MA": "Massachusetts", "ME": "Maine"} dict
```

Each of the above is an object.

Objects have

- a type (a particular object is said to be an instance of a type)
- an internal data representation
- a set of procedures for interacting with the object

Example: [1, 2, 3, 4]

- Type: `list`
- Internal data representation
  - int length L, an object array of size S ≥ L
    or, alternatively,
  - A linked list of individual cells
    <data, pointer to next cell>
- Procedures for manipulating lists
  - `l[i], l[i:j], l[i:j:k], +, *`
  - `len(), min(), max(), del l[i]`
  - `l.append(…), l.extend(…), l.count(…), l.index(…), l.insert(…), l.pop(…), l.remove(…), l.reverse(…), l.sort(…)`

The internal representation is meant to be private in the sense that users of the objects shouldn't rely on particular details of the implementation. Correct behavior may be compromised if you manipulate internal data directly.

Object-Oriented Programming (OOP)

- Everything is an object and has a type
- Objects are a data abstraction that encapsulate
  - Internal representation
  - Interface for interacting with the object
    - Defines behaviors, hides implementation
    - Attributes: data, methods (procedures)
- One can
  - Create new instances of objects (explicitly or using literals)
  - Destroy objects
    - explicitly using `del` or just "forget" about them
    - Python system will reclaim destroyed or inaccessible objects

Some languages have support for "data hiding" which prevents access to private attributes. Python does not… one is just expected to play by the rules!

Reclamation process is called "garbage collection"

Advantages of OOP

- Divide-and-conquer development
  - Implement and test behavior of each class separately
  - Increased modularity reduces complexity
- Classes make it easy to reuse code
  - Many Python modules define new classes
  - Each class has a separate namespace (no collision on function names)
  - Inheritance allows subclasses to redefine or extend a selected subset of a superclass' behavior