Objects

- Early programming languages did not provide ways to cluster data into coherent collections with well defined interfaces
- Meant that any piece of code to access any part of a data structure
- Lead to occurrence of hard to isolate bugs
- Much better if we can bundle data into packages together with procedures that work on them through well-defined interfaces

Objects

Python supports many different kinds of data:

```
1234  int      3.14159  float    "Hello"   str
[1, 2, 3, 5, 7, 11, 13]       list
{"CA": "California", "MA": "Massachusetts"}
                                  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 (primitive or composite)
- A set of procedures for interaction with the object

Example: [1,2,3,4]

- Type: `list`
- Internal data representation
  - int length L, an object array of size S >= L, or
  - A linked list of individual cells

```
    <data, pointer to next cell>
```

Example: [1,2,3,4]

- Type: `list`
- Internal data representation
  - int length L, an object array of size S >= L, or
  - A linked list of individual cells

    `<data, pointer to next cell>`

Internal representation is private – users of the objects should not rely on particular details of the implementation. Correct behavior may be compromised if you manipulate internal representation directly.

Example: [1,2,3,4]

- Type: `list`
- Internal data representation
  - int length L, an object array of size S >= L, or
  - 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(…)`

Internal representation is private – users of the objects should not rely on particular details of the implementation. Correct behavior may be compromised if you manipulate internal representation 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 object
    - Defines behaviors, hides implementation
    - Attributes: data, methods (procedures)

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 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 – called "garbage collection"

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!

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 environment (no collision on function names)
  - Inheritance allows subclasses to redefine or extend a selected subset of a superclass’ behavior