Example: a set of integers

- Create a new type to represent a set (or collection) of integers
  - Initially the set is empty
  - A particular integer appears only once in a set
    - This constraint, called a **representational invariant**, is enforced by the code in the methods.
- Internal data representation
  - Use a list to remember the elements of a set
- Interface
  - `insert(e)` – insert integer `e` into set if not there
  - `member(e)` – return `True` if integer `e` is in set, `False` else
  - `remove(e)` – remove integer `e` from set, error if not present

An implementation

```
class intSet(object):
    """An intSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""

    def __init__(self):
        """Create an empty set of integers"""
        self.vals = []

    def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)

    def __str__(self):
        """Returns a string representation of self"""
        self.vals.sort()
        return '{' + ','.join([str(e) for e in self.vals]) + '}'

  # other procedural attributes
```

An implementation

```
class intSet(object):
    """An intSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""”

  # other procedural attributes

     def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)

     def member(self, e):
        """Assumes e is an integer
        Returns True if e is in self, and False otherwise"""
        return e in self.vals
```

An implementation

```
class intSet(object):
    """An intSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""”

  # other procedural attributes

     def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)


    def remove(self, e):
        """Assumes e is an integer and removes e from self
        Raises ValueError if e is not in self"""
        try:
            self.vals.remove(e)
        except:
            raise ValueError(str(e) + ' not found')
```