Sets

Mathematical set: a collection of values, without duplicates or order

- Order does not matter
  \{1, 2, 3\} == \{3, 2, 1\}

- No duplicates
  \{3, 1, 4, 1, 5\} == \{5, 4, 3, 1\}

- For every data structure, ask:
  - How to create
  - How to query (look up) and perform other operations
    - (Can result in a new set, or in some other datatype)
  - How to modify

  Answer: [link to Python documentation]

Creating a Set

- Construct from a list:

  ```python
  odd = set([1, 3, 5])
  prime = set([2, 3, 5])
  empty = set([])
  ```

  Python always prints using this syntax above

Set Operations

```python
odd = set([1, 3, 5])
prime = set([2, 3, 5])
```

- membership $\in$ Python: in
  ```python
  4 in prime  ⇒ False
  ```

- union $\cup$ Python: |
  ```python
  odd | prime  ⇒ \{1, 2, 3, 5\}
  ```

- intersection $\cap$ Python: &
  ```python
  odd & prime  ⇒ \{3, 5\}
  ```

- difference $\setminus$ Python: -
  ```python
  odd - prime  ⇒ \{1\}
  ```

Think in terms of set operations, not in terms of iteration and element operations

- Shorter, clearer, less error-prone, faster

Although we can do iteration over sets:

```python
# iterates over items in arbitrary order
for item in myset:
...
```

But we cannot index into a set to access a specific element.
Modifying a Set

- **Add** one element to a set:
  
  ```python
  myset.add(newelt)
  myset = myset | set([newelt])
  ```

- **Remove** one element from a set:
  
  ```python
  myset.remove(elt)  # elt must be in myset or raises err
  myset.discard(elt) # never errs
  ```

What would this do?

- ```python
  myset = myset - set([newelt])
  ```

- **Choose** and remove some element from a set:
  
  ```python
  myset.pop()
  ```

List vs. Set Operations (1)

Find the common elements in both list1 and list2:

```python
out1 = []
for i in list2:
    if i in list1:
        out1.append(i)
```

OR

```python
out1 = [i for i in list2 if i in list1]
```

Find the common elements in both set1 and set2:

```python
set1 & set2
```

Much shorter, clearer, easier to write!

List vs. Set Operations (2)

Find the elements in either list1 or list2 (or both) (without duplicates):

```python
out2 = list(list1)  # make a copy
for i in list2:
    if i not in list1:
        out2.append(i)  # already in out2
```

OR

```python
out2 = list1+list2
for i in out1:
    out2.remove(i)  # common elements in both lists
```

Find the elements in either set1 or set2 (or both):

```python
set1 | set2
```
List vs. Set operations (3)

Find the elements in either list but not in both:
```
out3 = []
for i in list1+list2:
    if i not in list1 or i not in list2:
        out3.append(i)
```

Find the elements in either set but not in both:
```
set1 ^ set2  # symmetric difference
```

Not Every Value may be Placed in a Set - 1

• Set elements must be immutable values
  – int, float, bool, string, tuple
  – not: list, set, dictionary

• Goal: only set operations change the set
  – after “myset.add(x)”, x in myset ⇒ True
  – y in myset always evaluates to the same value
    Both conditions should hold until myset itself is changed

Not Every Value may be Placed in a Set - 2

• Mutable elements can violate these goals
  
  list1 = ["a", "b"]
  list2 = list1
  list3 = ["a", "b"]
  myset = { list1 }  ← Hypothetical;
  list1 in myset ⇒ True  actually illegal in Python
  list3 in myset ⇒ True
  list2.append("c")  ← not modifying myset "directly"
  list1 in myset ⇒ ???  modifying myset "indirectly" would
  list3 in myset ⇒ ???  lead to different results