Sets

BBM 101 - Introduction to Programming I

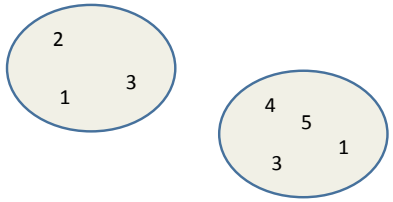
Hacettepe University
Fall 2015

Fuat Akal, Aykut Erdem, Erkut Erdem, Vahid Garousi

Slides based on material prepared by Ruth Anderson, Michael Ernst and Bill Howe in the course CSE 140 University of Washington

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: http://docs.python.org/2/library/stdtypes.html#set

Two Ways to Create a Set

1. Direct mathematical syntax:

```
odd = { 1, 3, 5 }
prime = { 2, 3, 5 }
```

Cannot express empty set: "`{}`" means something else ☹

2. Construct from a **list:**

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

Python always **prints** using this syntax above

Set Operations

```
odd = { 1, 3, 5 }
prime = { 2, 3, 5 }
```

- membership ∈ Python: `in` `4 in prime` ⇒ False
- union ∪ Python: `|` `odd | prime` ⇒ { 1, 2, 3, 5 }
- intersection ∩ Python: `&` `odd & prime` ⇒ { 3, 5 }
- difference \ or - 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:

```
# 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:

```
myset.add(newelt)
myset = myset | { newelt }
```

- **Remove** one element from a set:

```
myset.remove(elt)    # elt must be in myset or raises err
myset.discard(elt)   # never errs
myset = myset - { elt }
```

What would this do?

```
myset = myset - elt
```

- Choose and remove some element from a set:

```
myset.pop()
```


Practice with Sets

```
z = {5,6,7,8}
y = {1,2,3,"foo",1,5}
k = z & y
j = z | y
m = y - z
z.add(9)
```


List vs. Set Operations (1)

Find the common elements **in both** list1 and list2:

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

# We will learn about list comprehensions later
out1 = [i for i in list2 if i in list1]
```

Find the common elements in both set1 and set2:

```
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):

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

OR

```
out2 = list1+list2
for i in out1:              # out1 (from previous example), common
                            #  elements in both lists
   out2.remove(i)           # Remove common elements
```

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

```
set1 | set2
```

List vs. Set operations (3)

Find the elements in **either list but <u>not</u> 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
```


Not Every Value may be Placed in a Set

- 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
- Mutable elements can violate these goals

```
list1 = ["a", "b"]
list2 = list1
list3 = ["a", "b"]
myset = { list1 }          ⇐ Hypothetical; actually illegal in Python
list1 in myset ⇒ True
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
```