Sets

BBM 101 - Introduction to Programming I

Hacettepe University
Fall 2016

Fuat Akal, Aykut Erdem, Erkut Erdem

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

odd = set([ 1, 3, 5 ])
prime = set([ 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:

  ```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()
  ```
Practice with Sets

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

```
z: {8, 9, 5, 6, 7}
y: {1, 2, 3, 5, 'foo'}
k: {5}
j: {1, 2, 3, 5, 6, 7, 8, 'foo'}
m: {1, 2, 3, 'foo'}
```
List vs. Set Operations (1)

Find the common elements \textbf{in both} list1 and list2:

\begin{verbatim}
out1 = []
for i in list2:
    if i in list1:
        out1 .append(i)
\end{verbatim}

OR

\begin{verbatim}
out1 = [i for i in list2 if i in list1]
\end{verbatim}

Find the common elements in both set1 and set2:

\begin{verbatim}
set1 & set2
\end{verbatim}

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)  # don’t append elements
    # already in out2

OR

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

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:
```python
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:
```python
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

```python
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 ⇒ ???
list3 in myset ⇒ ???
```

modifying `myset` “indirectly” would lead to different results