Sets and Dictionaries (chapter 9)

Lists and Tuples (recap)

- Lists are mutable collections of objects (in brackets)

```
>>> countries = ["Australia", "Austria", "Bangladesh"]
>>> countries += ["Belize"]  # Modifies the original list
>>> countries[2]
'Bangladesh'
>>> "Austria" in countries
True
>>> countries.index("Bangladesh")
2
```

$O(n)$

- Tuples are immutable collections of objects (in parentheses)

```
>>> countries = ("Australia", "Austria", "Bangladesh")
>>> countries += ("Belize", )  # Creates a new tuple
>>> countries[3]
'Belize'
```

Sets

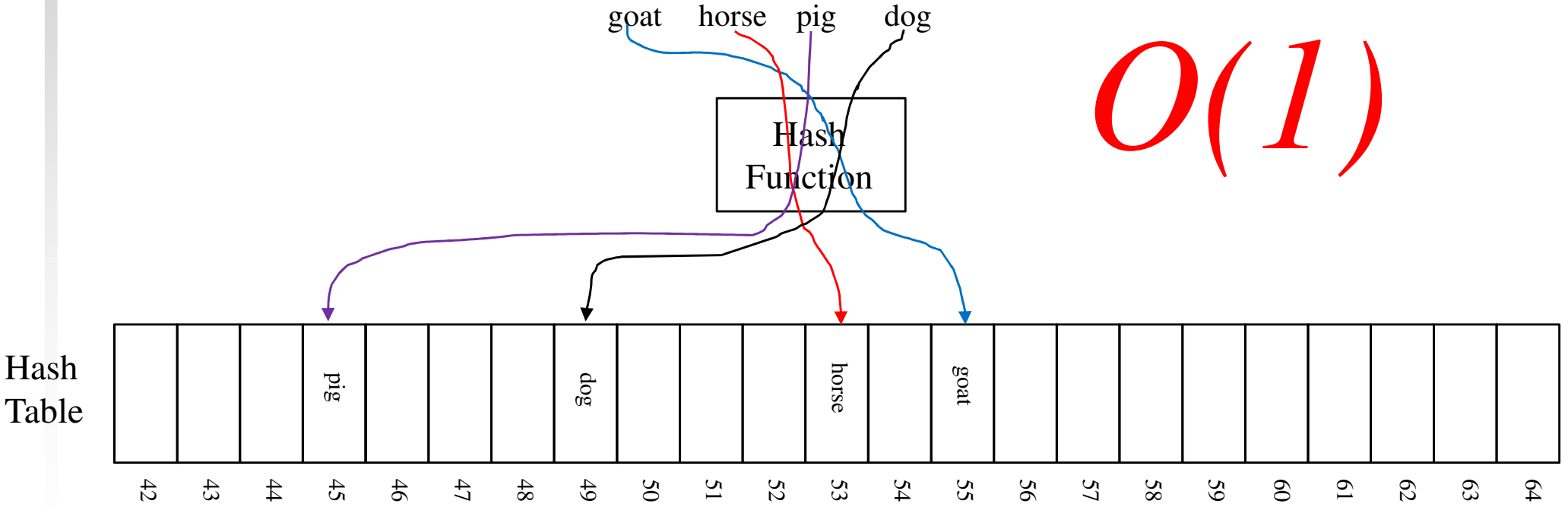
- Collections of unordered and distinct items
- Sets are mutable, but all items must be immutable
- Something is either in the set, or it's not
- Very fast due to hashing

- Useful for membership testing, removing duplicates, and general set math (intersection, union, difference, etc.)
- Created with `set` and zero or one iterable argument

```
>>> emptySet = set()
>>> farmyardAnimals = set(["goat", "dog", "pig"])
>>> early_letters = set("abcdefghi")
```

Hashing (aside)

- Fast look up method, used lots in computer science
- If objects change, their hash values changes
- Therefore immutable



Compare times…

```
big_num = 50000
big_list = range(big_num)

for i in big_list:
    i in big_list

print "done"
```

```
big_num = 50000
big_set = set(range(big_num))

for i in big_set:
    i in big_set

print "done"
```

Or a bit more formally…

```
def big_list_test(big_num):
        big_list = range(big_num)
        for i in big_list:
                i in big_list

def big_set_test(big_num):
        big_set = set(range(big_num))
        for i in big_set:
                i in big_set

# The following calls each function with an argument of 50000,
# and prints execution time.
if __name__ == '__main__':
    from timeit import Timer
    t = Timer("big_list_test(50000)", "from __main__ import big_list_test")
    print "List time:", t.timeit(number=1), "secs"
    t = Timer("big_set_test(50000)", "from __main__ import big_set_test")
    print "Set time:", t.timeit(number=1), "secs"
    print "DONE"
```

Sets Operations

```
>>> householdPets = set(["goldfish", "dog", "cat", "gerbil"])
>>> farmyardAnimals = set(["goat", "dog", "pig"])
>>> "pig" in farmyardAnimals
True
```

- `add` inserts an item into the set (mutator)

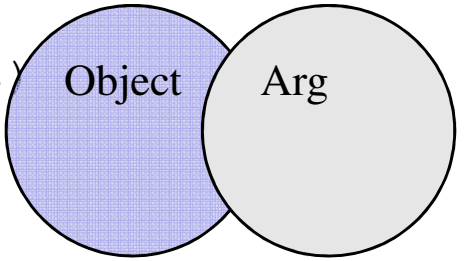
```
>>> farmyardAnimals.add("cow")
```

- `clear` empties the set (mutator)

```
>>> farmyardAnimals.clear()
```

- `difference` creates a new set containing items in the object, but not in the argument

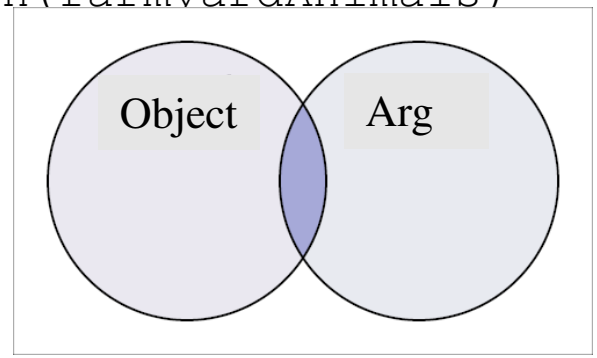
```
>>> print householdPets.difference(farmyardAnimals)
set(['goldfish', 'gerbil', 'cat'])
>>> print farmyardAnimals.difference(householdPets)
set(['goat', 'cow', 'pig'])
```



Sets Operations

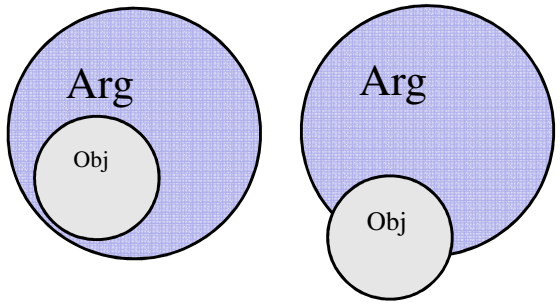
- `intersection` creates a new set containing items in both

```
>>> inBoth = householdPets.intersection(farmyardAnimals)
>>> print inBoth
set(['dog'])
```



- `issubset` are the object's items all in the argument?

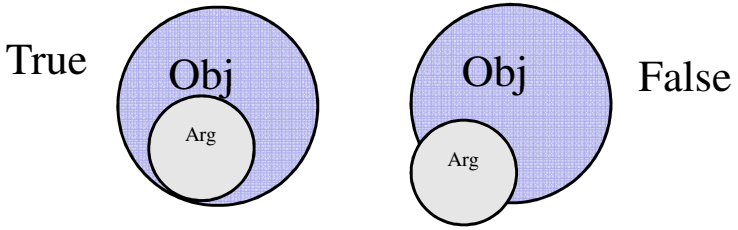
```
>>> andyPets = set(["cat", "dog"])
>>> andyPets.issubset(householdPets)
True
>>> saraPets = set(["snake", "dog"])
>>> saraPets.issubset(householdPets)
False
```



Sets Operations

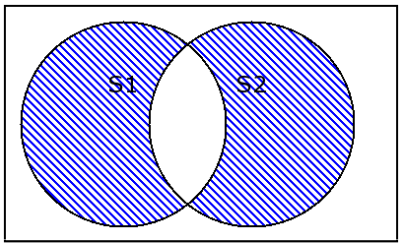
- `issuperset` does the object contain all the items in the argument?

```
>>> householdPets.issuperset(andyPets)
True
```

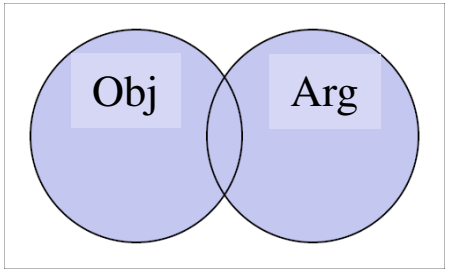


- `remove` removes an item from the set (mutator)

```
>>> andyPets.remove("dog")
>>> print andyPets
set(['cat'])
```



- `symmetric_difference` creates a set with items in exactly one set (either but not both)
- `union` creates a set with items in either set



Sets Operators

- Some operators provide alternative syntax for set methods

Method Call	Operator equivalent
`set1.difference(set2)`	set1 – set2
`set1.intersection(set2)`	set1 & set2
`set1.issubset(set2)`	set1 <= set2
`set1.issuperset(set2)`	set1 >= set2
`set1.union(set2)`	set1 | set2
`set1.symmetric_difference(set2)`	set1^set2

- Stylistic choice
- Note '+' operator is not defined for sets (potentially ambiguous; insertion, union, intersection?)

Sets of Sets

- Sets are mutable, but items must be immutable

```
>>> twoSets = set([householdPets, farmYardAnimals])
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    twoSets = set([householdPets, farmYardAnimals])
TypeError: unhashable type: 'set'
```

- Hmm… how to do sets of lists or sets?
- Use `frozensets`

```
>>> twoSets = set([frozenset(householdPets),
       frozenset(farmYardAnimals)])
>>>
```

Bird Observations (p182, sort of)

```
# setdict/birdwatching_mod.py

# retrieve filename from user
filename = raw_input("Enter file name: ")

# Find the different bird types observed.
birds = set()

infile = open(filename, 'r')
for line in infile:
    name = line.strip()
    birds.add(name)

# Print the birds.
for b in birds:
    print b
```

canada goose
canada goose
long-tailed jaeger
canada goose
snow goose
canada goose
canada goose
northern fulmar

Set Quiz!

```
>>> lows = set(range(4))
>>> print lows                  # output?
>>> print 4 in lows             # output?
>>> highs = set(range(5, 10))
>>> print highs                 # output?
>>> print 5 in highs            # output?
>>> lows += set([5])            # Oops!
>>> lows.add(5)
>>> print lows.intersection(highs)            # output?
>>> print highs.symmetric_difference(lows)    # output
>>> print lows.union(highs)                   # output?
>>> lows_n_highs = set([lows, highs])         # Oops!
>>> f_lows = frozenset(lows)
>>> f_highs = frozenset(highs)
>>> lows_n_highs = set([f_lows, f_highs])
>>> print f_lows in lows_n_highs              # output?
>>> print 4 in lows_n_highs                   # output?
>>> f_lows.add(5)                             # Oops!
```

Removing list duplicates via sets

```
>>> def remove_list_duplicates(our_list):
        set_list = set(our_list)
        return list(set_list)
>>> some_list = [1, 2, 2, 2, 3, 4, 3, 4, 5, 6, 6, 6, 7, 8]
>>> nl = remove_list_duplicates(some_list)
>>> print nl
[1, 2, 3, 4, 5, 6, 7, 8]

>>> # OR, shorter (and probably clearer)
>>> nl = list(set(some_list))
```

Sets, summary

- Collections of unordered and distinct items
- Sets are mutable, but all items must be immutable
- Something is either in the set, or it's not
- Very fast due to hashing – *O(1)*

- Useful for membership testing, removing duplicates, and general set math (intersection, union, difference, etc.)

Dictionaries

- Dictionaries define *key/value* pairs
- The keys form a set
  - Unordered
  - any key can appear once at most
  - keys must be immutable
- Values can change
- Curly braces { }, colons, and commas

```
>>> birds = {'canada goose' : 3, 'northern fulmar': 1}
>>> birds['northern fulmar']
1
```

- Type name is 'dict'

Dictionaries – Basics

- Accessing a non-existent key is an error

```
>>> birds['puffin']              # OOPS!
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
KeyError: 'puffin'
```

- Is the key in the dictionary? Use `in`

```
>>> if 'puffin' in birds:
      print "Puffins have been seen"
```

- Adding a key/value pair, or reassigning a value

```
>>> birds['puffin'] = 42
```

- Deleting a key (and its value). Use `del`

```
>>> del birds['puffin']
```

Dictionaries – Looping

- Dictionaries are unordered collections

```
>>> birds = {'canada goose' : 183, 'long-tailed jaeger'
: 71, 'snow goose' : 63, 'northern fulmar': 1}
>>> for x in birds:
        print x, birds[x]
northern fulmar 1
long-tailed jaeger 71
canada goose 183
snow goose 63
```

- Looping over a dictionary iterates over the unordered keys (unlike lists, which iterate over the ordered values)

Dictionaries – Methods

- `clear` Empties the dictionary

```
>>> d.clear()
```

- `get` Returns the value associated with the key, or an optional default if the key is not present

```
>>> birds = {'puffin':42, 'sparrow':14, 'wren':56}
>>> birds.get('puffin')
42
>>> birds.get('hawk', 99)
99
```

- `keys` Returns a list of the dictionary keys

```
>>> birds.keys()
['puffin', 'wren', 'sparrow']
```

- `items` Returns a list of key/value pairs

```
>>> birds.items()
[('puffin', 42), ('wren', 56), ('sparrow', 14)]
```

Dictionaries – Methods (cont.)

- `values` Returns a list of the dictionary values

```
>>> birds.values()
[42, 56, 14]
```

- `update` Add a set of key/value pairs to the dictionary

```
>>> morebirds = {'gull':7, 'hawk':14, 'pigeon':3246}
>>> birds.update(morebirds)
>>> birds
{'puffin': 42, 'sparrow': 14, 'gull': 7, 'wren': 56,
'hawk': 14, 'pigeon': 3246}
```

Dictionaries – Looping (cont.)

- You can easily loop over key/value pairs via multi-valued assignment

```
>>> for (key, value) in dictionary.items():
        # do something with the keys and values
```

- Or (more efficient, because no list is created)

```
>>> for (key, value) in dictionary.iteritems():
        # do something with the keys and values
```

Counting Birds Example I

```
# setdict/countbirds1_mod.py

# Dictionary of bird counts
bird_counts = {}

# retrieve filename from user
filename = raw_input("Enter the bird records file name: ")
infile = open(filename, 'r')
for line in infile:
        name = line.strip()
        if name in bird_counts:
                bird_counts[name] = bird_counts[name] + 1
        else:
                bird_counts[name] = 1

infile.close()
# Print.
for b in bird_counts:
        print b, bird_counts[b]
```

canada goose
canada goose
long-tailed jaeger
canada goose
snow goose
…

Counting Birds Example II

Shortened with `get`
and birds alphabetically sorted

```
# setdict/countbirds2_mod.py

bird_counts = {}
filename = raw_input("Enter the bird records file name: ")
infile = open(filename, 'r')
for line in infile:
        name = line.strip()
        bird_counts[name] = bird_counts.get(name, 0) + 1
infile.close()
# Print.
keys = bird_counts.keys()
keys.sort()
for b in keys:
        print b, bird_counts[b]
```

Counting Birds Example III

More readable print (book has wrong example on page 191)

```
# setdict/countbirds3_mod.py

bird_counts = {}
filename = raw_input("Enter the bird records file name: ")
infile = open(filename, 'r')
for line in infile:
        name = line.strip()
        bird_counts[name] = bird_counts.get(name, 0) + 1
infile.close()
# Print.
for b in sorted(bird_counts.keys()):
        print b, bird_counts[b]
```

Counting Birds Example IV

Inverting the dictionary

Sorting by value (frequency of bird observation)

```
# setdict/countbirds4_mod.py
bird_counts = {}
filename = raw_input("Enter the bird records file name: ")
infile = open(filename, 'r')
for line in infile:
        name = line.strip()
        bird_counts[name] = bird_counts.get(name, 0) + 1
infile.close()

#Invert the dictionary, creating a dictionary of the form
#   {1:['a','b']}, where 'a' and 'b' are birds
count_birds = {}
for (bird, count) in bird_counts.items():
        if count in count_birds:
                count_birds[count].append(bird)
        else:
                count_birds[count] = [bird]

# Print.
for count in sorted(count_birds.keys()):
        print count
        for bird in count_birds[count]:
                print ' ', bird
```

Dictionary Quiz!

```
>>> inventory = {'apples':430, 'pears':220, 'plums':540}

>>> inventory['pears']
>>> inventory[220]
>>> inventory['pears', 'plums']
>>> 'bananas' in inventory
>>> inventory.get('bananas', 1000)
>>> inventory.values()
>>> inventory['bananas'] = 1000
>>> inventory.keys()

>>> for fruit in sorted(inventory.keys()):
...     print fruit, inventory[fruit]
```

Dictionaries, summary

- Collections of unordered key/value pairs
- Each key must be of immutable type
- The set of keys in the dictionary can change
- Values can change
- Very fast due to hashing – *O(1)*

- Really useful for mapping between keys and values
- Much faster than lists (hashing, not searching)