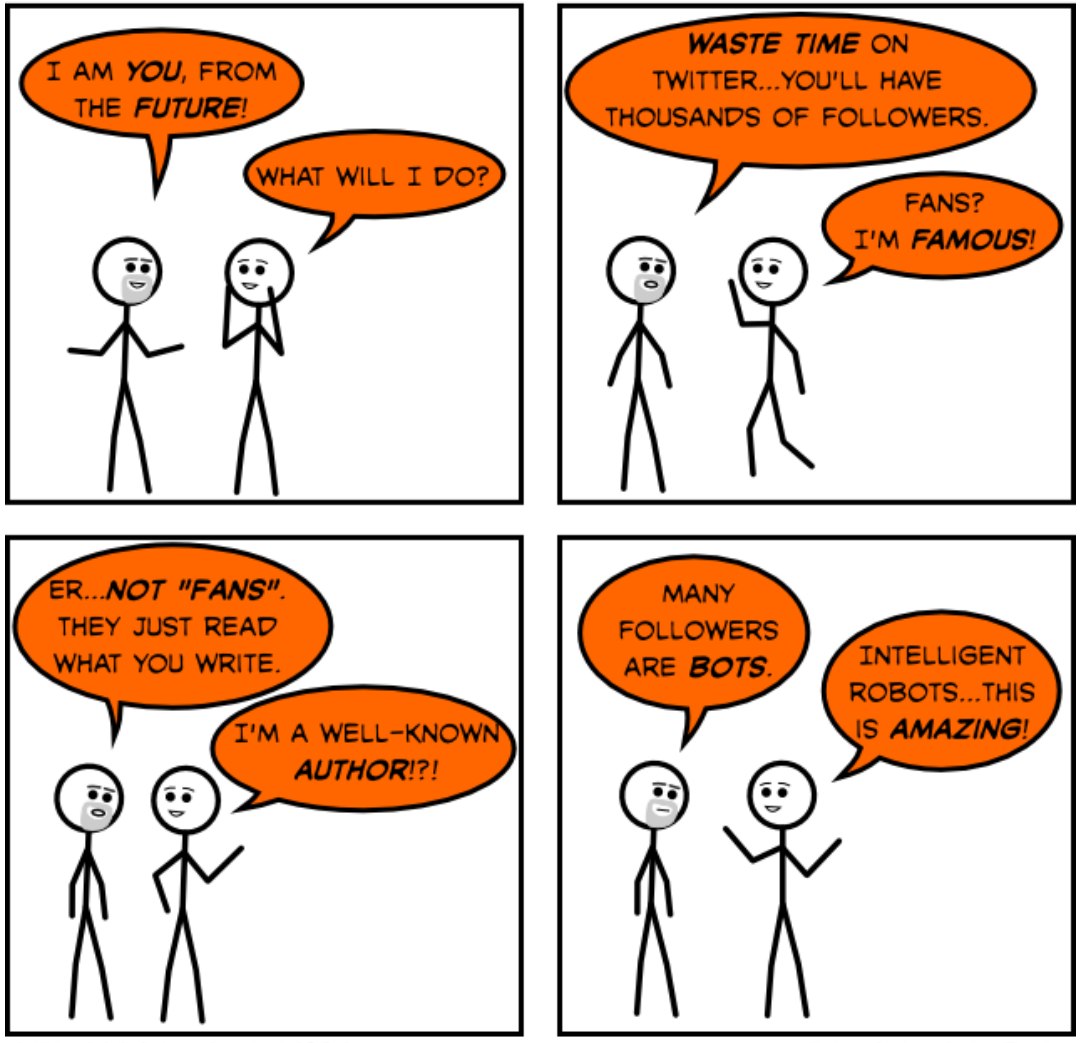
# Sets and Dictionaries

## Dealing with set and dictionary objects

D.S. Hwang

Department of Software Science
Dankook University

COMIC #32 - FUTURE YOU




HTTP://TWITTCH.COM/32/

Twittch by Eric Burke

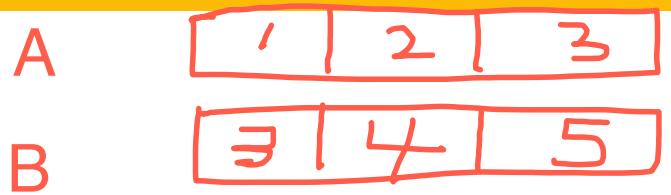
I LOVE My Computer Because My Friends Live In It

# Outline

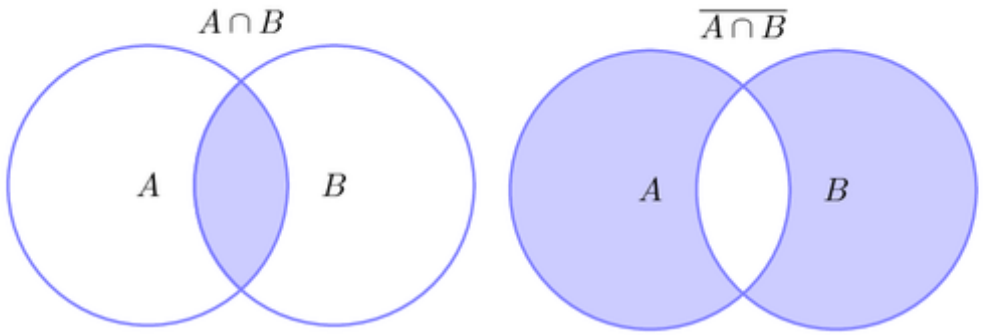
# Sets

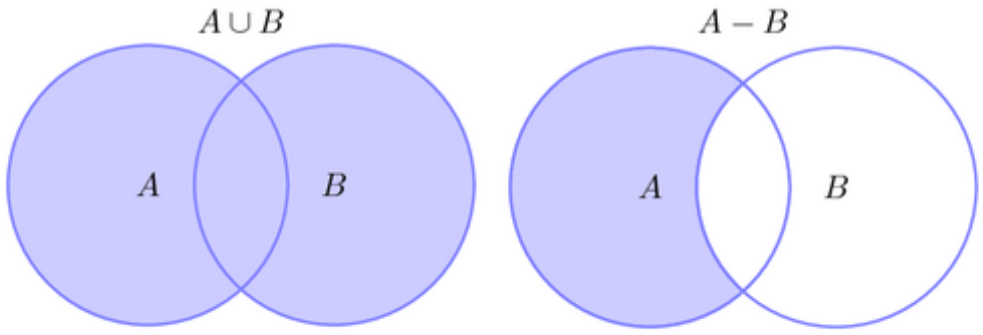
A={1,2,3}
B={3,4,5}

int A[4];
int B[6];




C
3



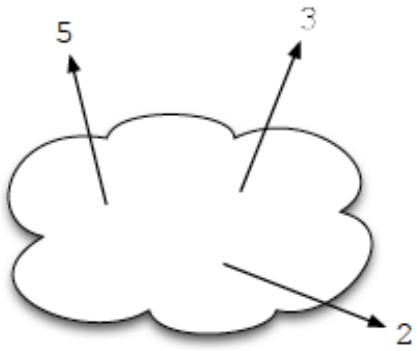
# Overview

- How to store multiple values in sequence?
  - list
  - tuple
- Two other kinds of collections
  - set
  - dictionary

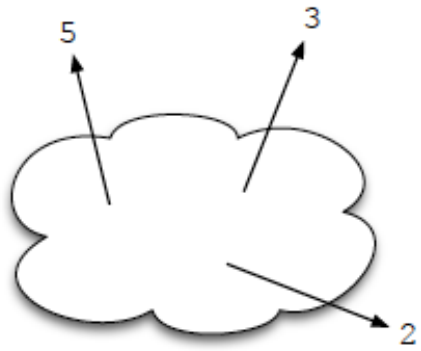
# Sets

Set

- an unordered collection of distinct items
- Items are not stored in any particular order.
- Sets are fundamental to mathematics and are built into modern versions of Python.




set([3, 5, 2])




set([2, 3, 5, 5, 2, 3])

# Sets

## Set operation

- `union`, `intersection`, and `difference` create a new set.
- `add`, `remove`, and `clear` modify the current set.

```
>>> ten = set(range(10))
>>> lows = set([0, 1, 2, 3, 4])
>>> odds = set([1, 3, 5, 7, 9])
>>> lows.add(9)
>>> lows
set([0, 1, 2, 3, 4, 9])
>>> lows.difference(odds)
set([0, 2, 4])
>>> lows.intersection(odds)
set([1, 3, 9])
>>> lows.issubset(ten)
True
```

lows is a subset of ten

$A \subseteq B$

```
>>> lows.issuperset(odds)
False
>>> lows.remove(0)
>>> lows
set([1, 2, 3, 4, 9])
>>> lows.symmetric_difference(odds)
set([2, 4, 5, 7])
>>> lows.union(odds)
set([1, 2, 3, 4, 5, 7, 9])
>>> lows.clear()
>>> lows
set([])
```

# Sets

## Set operations

| Method | Purpose | Example | Result |
|---|---|---|---|
| add | Adds an element to a set | lows.add(9) | None |
| clear | Removes all elements from a set | lows.clear() | None |
| difference | Creates a set with elements from one set, but not the other | lows.difference(odds) | set((0, 2, 4)) |
| intersection | Creates a set with elements that are in both sets | lows.intersection(odds) | set((1, 3)) |
| issubset | Asks are all of one set's elements contained in another? | lows.issubset(ten) | True |
| issuperset | Asks does one set contain all of another's elements? | lows.issuperset(odds) | False |
| remove | Removes an element from a set | lows.remove(0) | None |
| symmetric_difference | Creates a set with elements that are in exactly one set | lows.symmetric_difference(odds) | set((0, 2, 4, 5, 7, 9)) |
| union | Creates a set with elements that are in either set | lows.union(odds) | set((0, 1, 2, 3, 4, 5, 7, 9)) |

# Sets

Set storage

- Sets are stored in a data structure, called a hash table.
- Each time an item is added to a set, Python calculates a hash code for the item.

```
>>> help(hash)
Help on built-in function hash in module __builtin__:

hash(...)
    hash(object) -> integer

    Return a hash value for the object.  Two objects with the same
    value have the same hash value.  The reverse is not necessarily
    true, but likely.

>>> hash(123)
123
>>> hash('123') # a string
1911471187
```

# Dictionaries

Suppose we have several files recording observations of birds in the Canadian Arctic. We want to know which species we have seen.

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

# Dictionaries

Set data structure deals with this problem.

```
1  import sys
2  # Find the different bird types observed.
3  birds = set()
4  for filename in sys.argv[1:]:
5      infile = open(filename, 'r')
6      for line in infile:
7          name = line.strip()
8          birds.add(name)
9      infile.close()
10 # Print the birds.
11 for b in birds:
12     print( b )
```

# Dictionaries

We want to compute how often each kind of bird was seen.

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

# Dictionaries

A list of pairs can work for each bird.

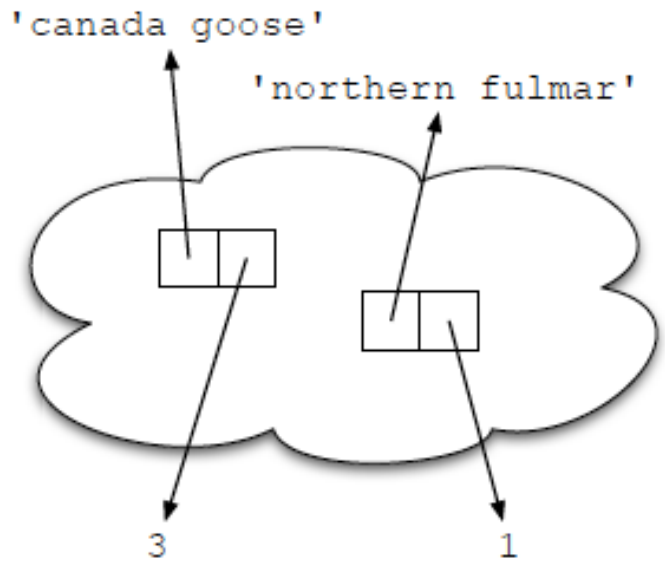
```
1  import sys
2  # Find all the birds.
3  birds = []
4  for filename in sys.argv[1:]:
5      infile = open(filename, 'r')
6      # For each bird, find its entry and increment the count.
7      for line in infile:
8          name = line.strip()
9          found = False
10         for entry in birds:
11             if entry[0] == name:
12                 entry[1] += 1; found = True
13         if not found:
14             birds.append([name, 1])
15     infile.close()
16 for (name, count) in birds:
17     print( name, count)
```

birds = [[name1, cnt1],...,[namek,cntk]]

# Dictionaries

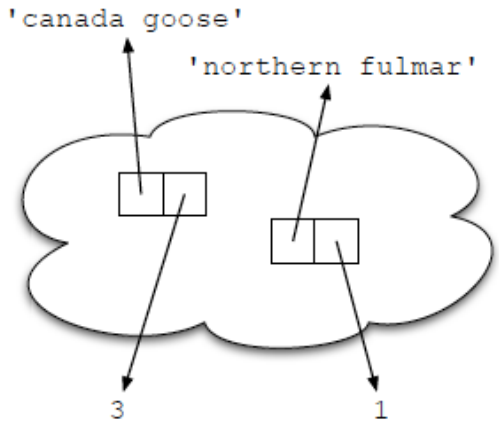
Dictionary or map

- an unordered mutable collection of key/value pairs
- Any particular key can appear at most once in a dictionary.
- Keys must be immutable.



# Dictionaries

```
>>> birds = {'canada goose' : 3, 'northern fulmar' : 1}
>>> birds['canada goose' ]
3
>>> birds = {'eagle' : 999, 'snow goose' : 33}
>>> if 'eagle' in birds:
...     print('eagles have been seen')
...
eagles have been seen
>>> del birds['eagle' ]
>>> if 'eagle' in birds:
...     print('oops: why are eagles still there?')
```



# Dictionaries

- Remove an item in a dictionary

```
>>> birds = {'snow goose' : 33, 'eagle' : 9}
>>> del birds['snow goose' ]
>>> birds
{'eagle' : 9}
```

- Loop over a dictionary

```
>>> 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
```

# Dictionaries

A dictionary can work much more easily than a list of pairs for each bird.

```
1  import sys
2  # Count all the birds.
3  count = {}
4      for filename in sys.argv[1:]:
5          infile = open(filename, 'r')
6          for line in infile:
7              name = line.strip()
8              if name in count:
9                  count[name] = count[name] + 1
10             else:
11                 count[name] = 1
12         infile.close()
13 # Print.
14 for b in count:
15     print (b, count[b])
```

dict()

# Dictionaries

count['canda goose']
count.get('canada goose')

Dictionary methods

| | |
|---|---|
| `keys` | returns the list of the dictionary's keys. |
| `values` | returns the list the dictionary's values. |
| `get` | returns the value associated with a key or some user-specified value if the key isn't in the dictionary. |
| `update` | updates keys and values from one dictionary into another. |
| `clear` | erases the dictionary's contents. |
| `items` | returns a list of (key, value) pairs. |

# Dictionaries

We can design the solution of the given problem a bit using the method `dict.get`.

```
import sys
# Count all the birds.
count = {}
for filename in sys.argv[1:]:
    infile = open(filename, 'r' )
    for line in infile:
        name = line.strip()
        count[name] = count.get(name, 0) + 1
    infile.close()# Print.
for b in count:
print( b, count[b])
```

# Dictionaries

We can get the dictionary's keys as a list, sort that list alphabetically.

```
1  import sys
2  # Count all the birds.
3  count = {}
4  for filename in sys.argv[1:]:
5      infile = open(filename, 'r')
6      for line in infile:
7          name = line.strip()
8          count[name] = count.get(name, 0) + 1
9      infile.close()
10 # Invert the dictionary.
11 freq = {}
12 for (name, times) in count.items():
13     if times in freq:
14         freq[times].append(name)
15     else:
16         freq[times] = [name]
17 # Print.
18 for key in sorted(freq):
19     print(key)
20     for name in freq[key]:
21         print(' ', name)
```

# Inverting a Dictionary

Invert the dictionary

- use the values as keys and the keys as values
- There's no guarantee that the values are unique. So we have to handle *collisions*.

# Inverting a Dictionary

```
1  import sys
2  # Count all the birds.
3  count = {}
4  for filename in sys.argv[1:]:
5      infile = open(filename, 'r')
6      for line in infile:
7          name = line.strip()
8          count[name] = count.get(name, 0) + 1
9      infile.close()
10 # Invert the dictionary.
11 freq = {}
12 for (name, times) in count.items():
13     if times in freq:
14         freq[times].append(name)
15     else:
16         freq[times] = [name]
17 # Print.
18 for key in sorted(freq):
19     print(key)
20     for name in freq[key]:
21         print ('  ', name)
```

# Summary

- Sets are used in Python to store unordered collections of unique values.
- Sets are stored in hash tables to make lookup efficient.
- Dictionaries are used to store unordered collections of key/value pairs.
- Looking things up in sets and dictionaries is much faster than searching through lists.

# Problem 1

## Homework

Design and implement a function that takes a set of integers as its input argument and returns a set of those integers that occur two or more times in the list. The input integers are generated randomly in $[1, 20]$.

# Problem 1

## Homework

Design and implement a function that takes a set of integers as its input argument and returns a set of those integers that occur two or more times in the list. The input integers are generated randomly in $[1, 20]$.

# Problem 2

## Homework

The keys in a dictionary are guaranteed to be unique, but the values are not. Write a function that takes a single dictionary as an argument and returns the number of distinct values it contains. For example, given the input `{'red': 1, 'green': 1, 'blue': 2}`, your function should return 2.

- Design and implement a function called `count_values1` that takes a single dictionary as an argument and returns the number of distinct values it contains.
- Based on set object, design and implement a function called `count_values2` that can do the same work.

# Problem 3 I

## Homework

A *sparse vector* is a vector whose entries are almost all zero, like `[1, 0, 0, 0, 0, 0, 3, 0, 0, 0]`. Storing all those zeros in a list wastes memory, so programmers often use dictionaries instead to keep track of just the nonzero entries. For example, the vector shown earlier would be represented as `{0:1, 6:3}`, because the vector it is meant to represent has the value 1 at index 0 and the value 3 at index 6.

{1:2, 6:2}

1. Design and implement a function called `normal_to_sparse` that converts a normal vector to its sparse vector.
2. Design and implement a function called `change_sign` that takes a sparse vector and returns its negative vector.
3. Design and implement a function called `add_vector` that takes two sparse vectors, adds them and returns a sparse vector representing their sum.

# Problem 3 II

## Homework

4. Extend the function `minus_vector` that takes two sparse vectors, subtracts them and return the result.

minus_vector(A,B) = A-B
= A + change_sign(B)
=add_vector(A,change_sign(B))