Sets and Dictionaries

Nanotech Example

How many molecules of different kinds can we make using the atoms in our warehouse?

How many molecules of different kinds can we make using the atoms in our warehouse?

```
# Molecular formula file

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

How many molecules of different kinds can we make using the atoms in our warehouse?

```
# Molecular formula file

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

```
# Atom inventory file

He 1
H 4
O 3
```

How many molecules of different kinds can we make using the atoms in our warehouse?

```
# Molecular formula file

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

```
# Atom inventory file

He 1
H 4
O 3
```

Now have all the tools we need

Natural to represent inventory as dictionary

Natural to represent inventory as dictionary

Keys: atomic symbols

Natural to represent inventory as dictionary

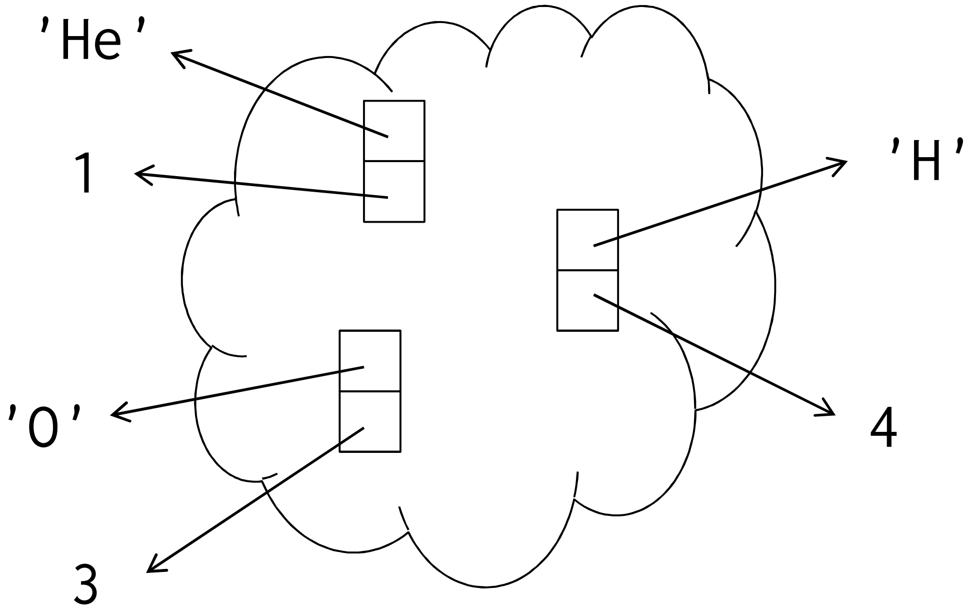
Keys: atomic symbols

Values: number of atoms available

Natural to represent inventory as dictionary

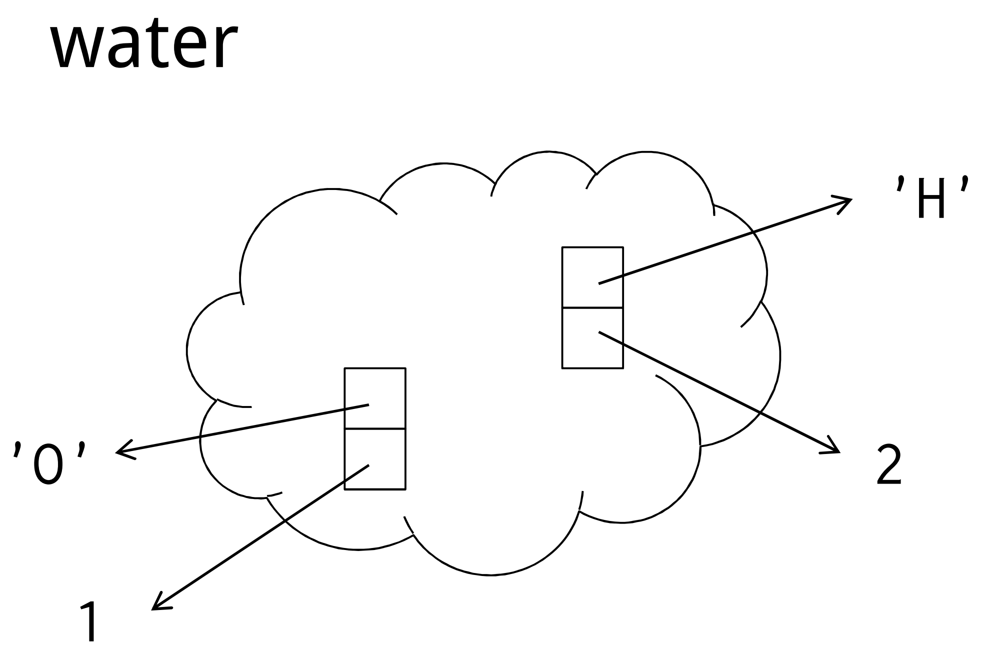
Keys: atomic symbols

Values: number of atoms available



Represent individual molecules the same way

Represent individual molecules the same way



Store formulas as a dictionary of dictionaries

Store formulas as a dictionary of dictionaries

Keys: molecule names

Store formulas as a dictionary of dictionaries

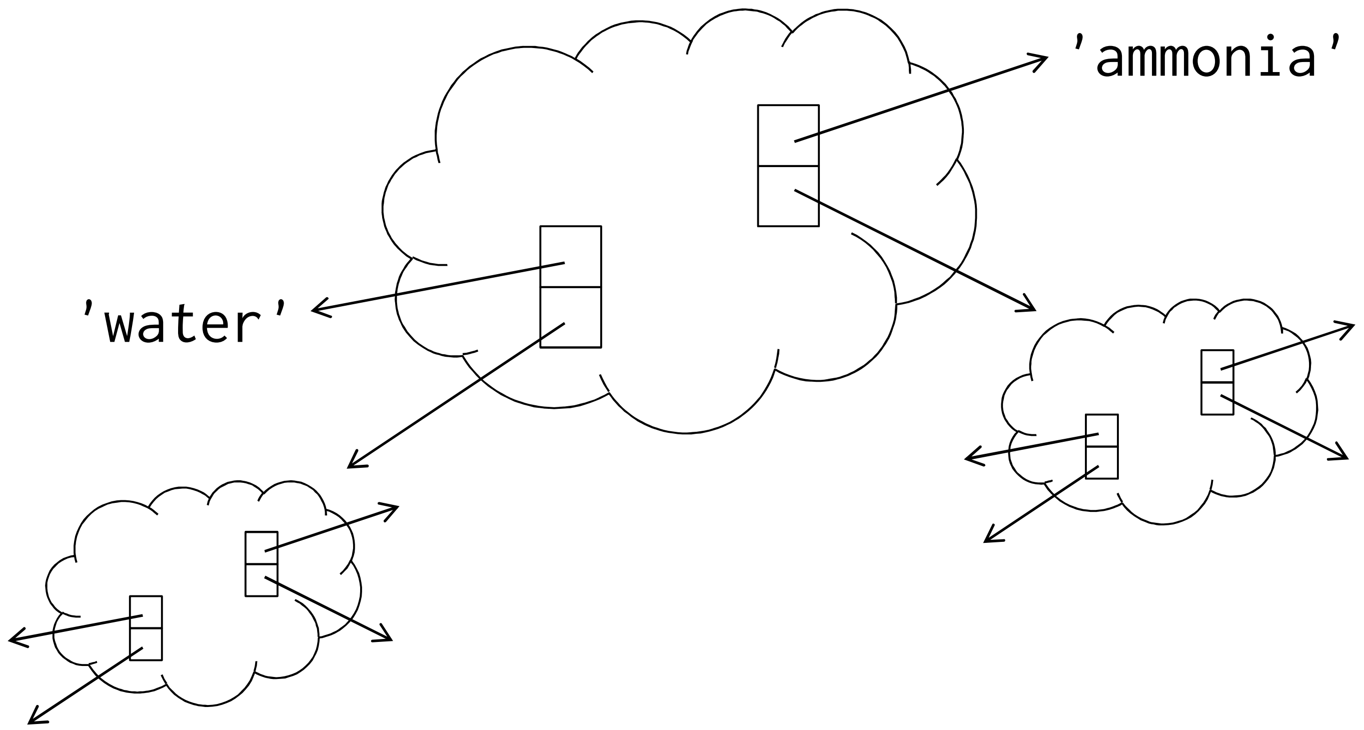
Keys: molecule names

Values: dictionaries of formulas

Store formulas as a dictionary of dictionaries

Keys: molecule names

Values: dictionaries of formulas



Number of molecules that can be made is:

$$\min_{\text{atom} \in \text{formula}} \frac{\text{available[atom]}}{\text{required[atom]}}$$

Number of molecules that can be made is:

$$\min_{\text{atom} \in \text{formula}} \frac{\text{available[atom]}}{\text{required[atom]}}$$

If `atom` not in `available`, its count is implicitly 0

Number of molecules that can be made is:

$$\min_{atom \in formula} \frac{available[atom]}{required[atom]}$$

If `atom` not in `available`, its count is implicitly 0

Store results in yet another dictionary

Number of molecules that can be made is:

$$\min_{\text{atom} \in \text{formula}} \frac{\text{available[atom]}}{\text{required[atom]}}$$

If `atom` not in `available`, its count is implicitly 0

Store results in yet another dictionary

Keys: molecule names

Number of molecules that can be made is:

$$\min_{\text{atom} \in \text{formula}} \frac{\text{available[atom]}}{\text{required[atom]}}$$

If `atom` not in `available`, its count is implicitly 0

Store results in yet another dictionary

Keys: molecule names

Values: counts of how many can be made

```python
'''Calculate how many molecules of each type can be made
with the atoms on hand.'''

import sys

if __name__ == '__main__':
  inventory = read_inventory(sys.argv[1])
  formulas = read_formulas(sys.argv[2])
  counts = calculate_counts(inventory, formulas)
  show_counts(counts)
```

```
def read_inventory(filename):
  '''Read inventory of available atoms.'''

  result = {}
  for line in read_lines(filename):
    name, count = line.split(' ')
    result[name] = int(count)

  return result
```

```
def read_lines(filename):
  '''Read lines from file, stripping out
  blank lines and comments.'''

  reader = open(filename, 'r')
  lines = []
  for line in reader:
    line = line.split('#')[0].strip()
    if line:
      lines.append(line)
  reader.close()

  return lines
```

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

Storing results in dictionary

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

$\leftarrow$ For each interesting line in the input file...

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

$\leftarrow$ Separate the molecule name and the formula

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

Separate the atoms and their counts

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

Loop over pairs of atoms and counts

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

Store the count as an integer with the atomic symbol as key

```
def read_formulas(filename):
  '''Read molecular formulas from file.'''

  result = {}
  for line in read_lines(filename):

    name, atoms = line.split(':')
    name = name.strip()

    atoms = atoms.strip().split(' ')
    formula = {}
    for i in range(0, len(atoms), 2):
      formula[atoms[i]] = int(atoms[i+1])

    result[name] = formula

  return result
```

And store the molecule in the main dictionary

```
def calculate_counts(inventory, formulas):
  '''Calculate how many of each molecule
  can be made with inventory.'''

  counts = {}
  for name in formulas:
    counts[name] = dict_divide(inventory,
                               formulas[name])

  return counts
```

```
def calculate_counts(inventory, formulas):
  '''Calculate how many of each molecule
  can be made with inventory.'''

  counts = {}
  for name in formulas:
    counts[name] = dict_divide(inventory,
                                  formulas[name])

  return counts
```

Sub-dictionary holding atom counts for a particular molecule

```
def calculate_counts(inventory, formulas):
  '''Calculate how many of each molecule
  can be made with inventory.'''

  counts = {}
  for name in formulas:
    counts[name] = dict_divide(inventory,
                               formulas[name])

  return counts
```

Big functions: nothing is obviously wrong

```
def calculate_counts(inventory, formulas):
  '''Calculate how many of each molecule
  can be made with inventory.'''

  counts = {}
  for name in formulas:
    counts[name] = dict_divide(inventory,
                               formulas[name])

  return counts
```

Big functions: nothing is obviously wrong

Small functions: obviously, nothing is wrong

```
def dict_divide(inventory, molecule):
  '''Calculate how much of a single molecule
  can be made with inventory.'''

  number = None
  for atom in molecule:
    required = molecule[atom]
    available = inventory.get(atom, 0)
    limit = available / required
    if (number is None) or (limit < number):
      number = limit

  return number
```

```
def dict_divide(inventory, molecule):
  '''Calculate how much of a single molecule
  can be made with inventory.'''

  number = None
  for atom in molecule:
    required = molecule[atom]
    available = inventory.get(atom, 0)
    limit = available / required
    if (number is None) or (limit < number):
      number = limit

  return number
```

Identical format:

keys are atoms,

values are counts

```
def dict_divide(inventory, molecule):
  '''Calculate how much of a single molecule
  can be made with inventory.'''

  number = None
  for atom in molecule:
    required = molecule[atom]
    available = inventory.get(atom, 0)
    limit = available / required
    if (number is None) or (limit < number):
      number = limit

  return number
```

Common pattern: None means "uninitialized", so initialize the first time through the loop

```
def dict_divide(inventory, molecule):
  '''Calculate how much of a single molecule
  can be made with inventory.'''

  number = None
  for atom in molecule:
    required = molecule[atom]
    available = inventory.get(atom, 0)
    limit = available / required
    if (number is None) or (limit < number):
      number = limit

  return number
```

Common pattern:

stored value or default

```
def dict_divide(inventory, molecule):
  '''Calculate how much of a single molecule
  can be made with inventory.'''

  number = None
  for atom in molecule:
    required = molecule[atom]
    available = inventory.get(atom, 0)
    limit = available / required
    if (number is None) or (limit < number):
      number = limit

  return number
```

Common pattern:

find minimum of

calculated values

```
def show_counts(counts):
  '''Show how many of each molecule can be made.'''

  counts = invert_dict(counts)
  for key in sorted(counts.keys(), reverse=True):
    for name in sorted(counts[key]):
      print key, name
```

```
def show_counts(counts):
  '''Show how many of each molecule can be made.'''

  counts = invert_dict(counts)
  for key in sorted(counts.keys(), reverse=True):
    for name in sorted(counts[key]):
      print key, name
```

Reverse to get greatest first

```
def invert_dict(src):
  '''Invert a dictionary, returning value->set{key}.'''

  dst = {}
  for key in src:
    value = src[key]
    if value not in dst:
      dst[value] = set()
    dst[value].add(key)

  return dst
```

```
def invert_dict(src):
  '''Invert a dictionary, returning value->set{key}.'''

  dst = {}
  for key in src:
    value = src[key]
    if value not in dst:
      dst[value] = set()
    dst[value].add(key)

  return dst
```

Common pattern:
make sure there's a
collection, then add

```
def show_counts(counts):
  '''Show how many of each molecule can be made.'''

  counts = invert_dict(counts)
  for key in sorted(counts.keys(), reverse=True):
    if key > 0:
      for name in sorted(counts[key]):
        print key, name
```

Go back and only show molecules that can actually be made

```
# inventory-00.txt
```

```
# formulas-01.txt

helium : He 1
```

No output, which is correct.

```
# inventory-01.txt
He 1
```

```
# formulas-01.txt

helium : He 1
```

1 helium

```
# inventory-02.txt
He 1
H 4
```

```
# formulas-01.txt

helium : He 1
```

```
1 helium
```

```
# inventory-02.txt
He 1
H 4
```

```
# formulas-02.txt

helium : He 1
water : H 2 O 1
```

```
1 helium
```

```
# inventory-02.txt
He 1
H 4
```

```
# formulas-03.txt

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

2 hydrogen

1 helium

```
# inventory-03.txt
He 1
H 4
O 3
```

```
# formulas-03.txt

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

2 hydrogen

2 water

1 helium

```
# inventory-03.txt
He 1
H 4
O 3
```

```
# formulas-03.txt

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

```
2 hydrogen

2 water

1 helium
```

Looks good...

```
# inventory-03.txt
He 1
H 4
O 3
```

```
# formulas-03.txt

helium : He 1
water : H 2 O 1
hydrogen : H 2
```

```
2 hydrogen

2 water

1 helium
```

Looks good...

Code is *much* simpler than it would be using lists of pairs created by

Greg Wilson

June 2010