Containers

Dictionary

Container data types

- list
- tuple
- Dictionaries
  - dict
- Sets
  - set

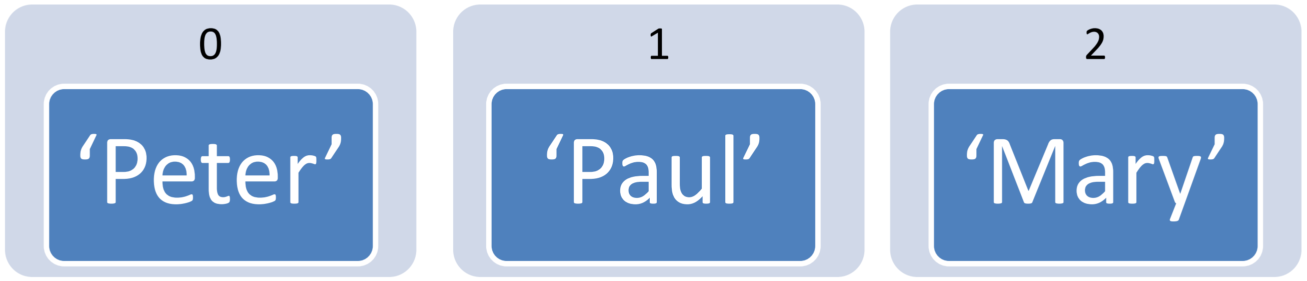
List

- A list is an indexed, ordered, container
- Each element of the list has a “label” (its index) that is automatically assigned

```
>>> myList = [‘Peter’,’Paul,’Mary’]
```

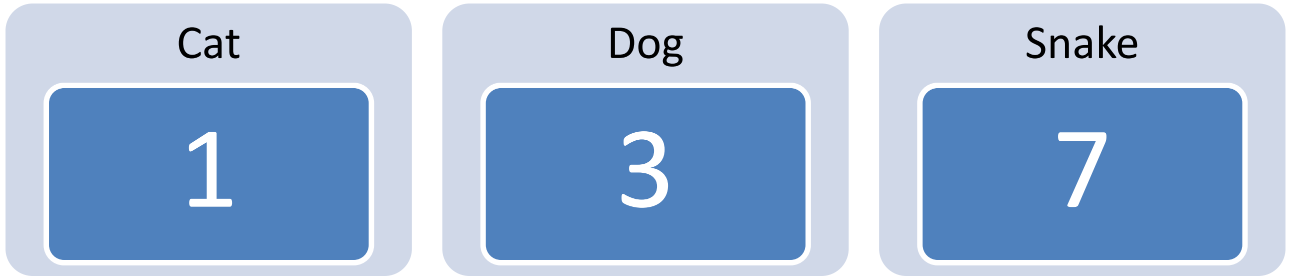
- Elements can be retrieved using the indexing operator

```
>>> myList[2]
Mary
```



Number of Animals

- Two lists
  - Animals
  - Values
- Needed to keep them in synch
- Would have been nice to be able to have a way of using the animals name as a **key** (label) for a box containing the **value**



Dictionaries

The dictionary class `dict` is designed to address exactly these situations

key	value
‘dog'	3
‘cat'	1
‘snake'	7

```
>>> zoo = {
          ‘cat’: 1,
          ‘dog': 3,
          ‘snake’:7}
>>> zoo[‘cat']
1
>>> zoo[‘snake’]
7
```

A dictionary contains
(key, value) pairs

A key can be used as an index to access the corresponding value

Basic dictionary syntax

- Curly braces
- Colons between values and keys
- Commas between pairs key:value

{ key1:value1, key2:value2,…..}

- For ease of reading often entered as

```
{
    key1:value1,
    key2:value2,
    …
}
```

One more example

Goal: a container of employee records indexed by employee SS#

```
>>> employee[987654321]
['Yu', 'Tsun']
>>> employee[864209753]
['Anna', 'Karenina']
>>> employee[100010010]
['Hans', 'Castorp']
```

Problems:

- the range of SS#s is huge
- SS#s are not really integers

Solution: the dictionary class `dict`

key	value
'864-20-9753'	['Anna', 'Karenina']
'987-65-4321'	['Yu', 'Tsun']
'100-01-0010'	['Hans', 'Castorp']

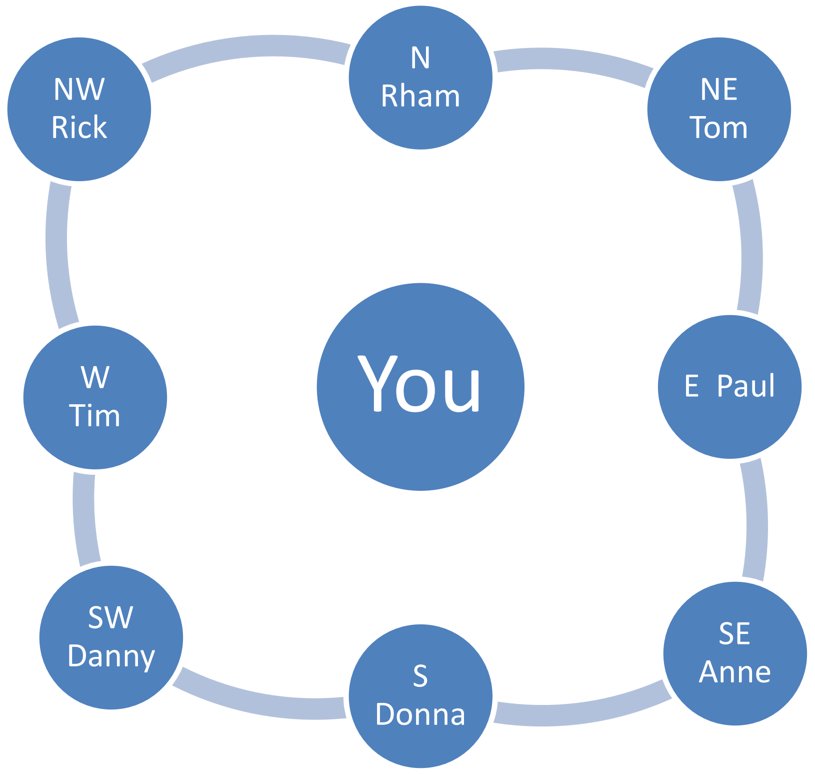
```
>>> employee = {
        '864-20-9753': ['Anna', 'Karenina'],
        '987-65-4321': ['Yu', 'Tsun'],
        '100-01-0010': ['Hans', 'Castorp']}
>>> employee['987-65-4321']
['Yu', 'Tsun']
>>> employee['864-20-9753']
['Anna', 'Karenina']
```

A dictionary contains (key, value) pairs

A key can be used as an index to access the corresponding value

Exercise

- Create a dictionary called neighbors that maps the location of your neighbors relative to you (see image below) to their names
- Enter ‘No one’ if you do not have a neighbor in a certain position



```
>>> neighbors[‘NE’]
Tom
```

BASIC SYNTAX
Curly braces

Colons between values and keys

Commas between pairs key:value

{ key1:value1, key2:value2,.....}

Properties of Dictionaries

Properties of dictionaries

Dictionaries are not ordered

Dictionaries are mutable

- new (key,value) pairs can be added
- the value corresponding to a key can be modified

The empty dictionary is {}

```
>>> employee = {
         '864-20-9753': ['Anna',
'Karenina'],
         '987-65-4321': ['Yu', 'Tsun'],
         '100-01-0010': ['Hans', 'Castorp']}
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun']}
>>> employee['123-45-6789'] = 'Holden
Cafield'
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun'], '123-45-6789':
'Holden Cafield'}
>>> employee['123-45-6789'] = 'Holden
Caulfield'
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-
20-9753': ['Anna', 'Karenina'], '987-65-
4321': ['Yu', 'Tsun'], '123-45-6789':
'Holden Caulfield’}
```

Keys and values: what data type?

Dictionary keys must be immutable

- strings
- ints
- tuples
- …

```
>>> employee = {[1,2]:1, [2,3]:3}
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    employee = {[1,2]:1, [2,3]:3}
TypeError: unhashable type: 'list'
```

Dictionary values can be anything

Dictionary operators

Class `dict` supports some of the same operators as class `list`:

- `indexing []`
- `in`
  - Used on KEYS!!
- `len`

Class dict does not support all the operators that class list supports
`+` and `*` for example

```
>>> days = {'Mo':1, 'Tu':2, 'W':3}
>>> days['Mo']
1
>>> days['Th'] = 4
>>> days
{'Th': 4, 'Tu': 2, 'Mo': 1, 'W': 3}
>>> 'Fr' in days
False
>>> len(days)
4
>>> days['Th'] = 5
>>> days
{'Th': 5, 'Tu': 2, 'Mo': 1, 'W': 3}
>>> days['Th'] = 4
>>> days
{'Th': 4, 'Tu': 2, 'Mo': 1, 'W': 3}
>>> 'Fr' in days
False
>>> len(days)
4
```

Exercise

- Create a dictionary called myDict that contains the following key-values pairs:
- A:123, B:234, C:1, D:45
- After creating the dictionary, add dynamically a new key-value pair E:34
- Retrieve the value corresponding to key B
- Check if key E is in the dictionary
- Compute the length of the dictionary

Dictionary Methods

Iterating on a Dictionary

Dictionary methods

Operation	Explanation
`d.items()`	Returns a view of the (key, value) pairs in d
`d.keys()`	Returns a view of the keys of d
`d.pop(key)`	Removes the (key, value) pair with key key from d and returns the value
`d.update(d2)`	Adds the (key, value) pairs of dictionary d2 to d
`d.values()`	Returns a view of the values of d

The containers returned by `d.items()`, `d.keys()`, and `d.values()` (called views) can be iterated over

```
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 4, 'W': 3}
>>> days.pop('Tu')
2
>>> days
{'Mo': 1, 'Th': 4, 'W': 3}
>>> days2 = {'Tu':2, 'Fr':5}
>>> days.update(days2)
>>> days
{'Fr': 5, 'W': 3, 'Th': 4, 'Mo': 1,
'Tu': 2}
>>> days.items()
dict_items([('Fr', 5), ('W', 3), ('Th',
4), ('Mo', 1), ('Tu', 2)])
>>> days.keys()
dict_keys(['Fr', 'W', 'Th', 'Mo', 'Tu'])
>>> >>> vals = days.values()
>>> vals
dict_values([5, 3, 4, 1, 2])
>>>
```

Iterating on a dictionary

```
>>> myDicnry={'a':1,'b':2,'c':324}
>>> for key in myDicnry.keys():
            myDicnry[key] +=1
>>> myDicnry
{'a': 2, 'c': 325, 'b': 3}
```

Iterating on a dictionary

```
>>> myDicnry={'a':1,'b':2,'c':324}
>>> for value in myDicnry.values():
                print(value, end=‘ ‘)
>>> 1,2,324
```

Iterating on a dictionary

- Careful:
  - You cannot alter values by iterating on them
- `>>> myDicnry={'a':1,'b':2,'c':324}`

```
>>> for value in myDicnry.values():
            value +=1
>>> myDicnry
{'a': 1, 'b': 2, 'c': 324}
```

Iterating on a dictionary

```
>>> myDicnry={'a':1,'b':2,'c':324}
>>> for i in myDicnry.items():
            print(i, end=‘ ‘)
>>> ('a', 1) (‘b', 2) (‘c', 324)
```

- What kind of objects did we get back?
- Each item (key, value) is a TUPLE
- Hence, IMMUTABLE in the view!!

Iterating on a dictionary

- As usual Python lets us “get away” with a fairly natural way of doing things…

```
>>> myDicnry={'a':1,'b':2,'c':324}
>>> for item in myDicnry:
            myDicnry[item] +=1
>>> myDicnry
{'a': 2,  'b': 3, 'c': 325, }
```

Exercise

- Create a dictionary called myDict that contains the following key-values pairs:
- A:123, B:234, C:1, D:45, E:34
- Write a loop that increments each value in the dictionary by 5

Example

- Using the scores.csv file, read file and store each student with grade for look- up