

## Advanced Data Structures: Sets and Dictionaries

CSE 231, Bill Punch

### More Data Structures

- We have seen the list data structure and what it can be used for
- We will now examine two more advanced data structures, the Set and the Dictionary
- In particular, the dictionary is an important, very useful part of python, as well as generally useful to solve many problems.



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Dictionaries

### What is a dictionary?

- In data structure terms, a dictionary is better termed an *associative array* or *associative list*
- You can think of it as a list of pairs, where the first element of the pair, the *key*, is used to retrieve the second element, the *value*



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

### Key Value pairs

- The key acts as a “lookup” to find the associated value.
- Just like a dictionary, you look up a word by its spelling to find the associated definition
- A dictionary can be searched to locate the value associated with a key



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

### Python Dictionary

- Use the {} marker to create a dictionary
  - Use the : marker to indicate key:value pairs
- ```
myDict = {'e':2.7183, 'pi':3.1416, 'h':
6.6261e-34}
print myDict
{'h': 6.6261399999999998e-034,
'pi': 3.1415999999999999,
'e': 2.7183000000000002}
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## keys and values

- Key must be immutable
  - strings, integers, tuples are fine
  - lists are NOT
- Value can be anything



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Access dictionary elements

Access requires [], but the *key* is the index!

```
myDict={}
    □ an empty dictionary
myDict['bill']=25
    □ added the pair 'bill':25
print myDict['bill']
    □ prints 25
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## again, common operators

Like others, dictionaries respond to these

- `len(myDict)`
  - number of key:value pairs in the dictionary
- `element in myDict`
  - boolean, is `element` a **key** in the dictionary
- `for key in myDict:`
  - iterates through the keys of a dictionary



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Lots of methods

- `myDict.items()` – all the key/value pairs
- `myDict.keys()` – all the keys
- `myDict.values()` – all the values
- `myDict.has_key(key)` or `key in myDict` does the key exist in the dictionary
- `myDict.clear()` – empty the dictionary
- `myDict.update(yourDict)` – for each key in `yourDict`, updates `myDict` with that key/value pair



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Dictionaries are iterable

```
for key in myDict:
    print key
    □ prints all the keys
for key,value in myDict.items():
    print key,value
    □ prints all the key/value pairs
for value in myDict.values():
    print value
    □ prints all the values
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Building dictionaries faster

- `zip` creates pairs from two parallel lists
  - `zip("abc",[1,2,3])` yields `[('a',1),('b',2),('c',3)]`
- That's good for building dictionaries. We call the `dict` function which takes a list of pairs to make a dictionary
  - `dict(zip("abc",[1,2,3]))` yields `{'a': 1, 'c': 3, 'b': 2}`



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Example: simple Dicts

## Example word Frequency

## Example: phoneBook

CSE 231, Bill Punch

## Lookup of functions

Look at the following

```
commandSet={"init":initFromFile,"lookup":lookup,
"add":addPair,"del":delPair,"print":printBook}

response = raw_input("Command:").strip()

commandSet[response]()
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Storing functions

- Once a function is defined, it is an object just like any other variable
- It can be stored as a value in a dictionary (indexed however you like)
- You can call the function by appending the () to the end of it.
- Very convenient!



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Sets

CSE 231, Bill Punch

## Sets, as in Mathematical Sets

- in mathematics, a set is a collection of objects, potentially of many different types
- in a set, no two elements are identical. That is, a set consists of elements each of which is unique compared to the other elements
- there is no order to the elements of a set
- a set with no elements is the empty set



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Creating a set

```
mySet = set("abcd")
```

- the "set" keyword creates a set
- the single argument that follows must be *iterable*, that is, something that can be walked through one item at a time with a `for`
- the result is a set data structure

```
print mySet
set(['a', 'c', 'b', 'd'])
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Diverse elements

- A set can consist of a mixture of different types of elements

```
mySet = set(['a', 1, 3.14159, True])
```

- as long as the single argument can be iterated through, you can make a set of it



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## no duplicates

- duplicates are automatically removed

```
mySet = set("aabbccdd")
print mySet
set(['a', 'c', 'b', 'd'])
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## common operators

Most data structures respond to these:

- `len(mySet)`
  - the number of elements in a set
- `element in mySet`
  - boolean indicating whether element is in the set
- `for element in mySet:`
  - iterate through the elements in mySet

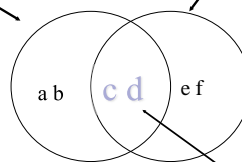


Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

## Set Ops, Intersection

```
mySet=set("abcd"); newSet=set("cdef")
```



```
mySet.intersection(newSet) returns set(['c', 'd'])
```



Michigan State University  
CSE 231, Fall 2009

Sets and Dictionaries

### Set Ops, Difference

```
mySet=set("abcd"); newSet=set("cdef")
```

```
mySet.difference(newSet) returns set(['a','b'])
```

Michigan State University  
CSE 231, Fall 2009  
Sets and Dictionaries

### Set Ops, Union

```
mySet=set("abcd"); newSet=set("cdef")
```

```
mySet.union(newSet) returns set(['a','b','c','d','e','f'])
```

Michigan State University  
CSE 231, Fall 2009  
Sets and Dictionaries

### Set Ops, symmetricDifference

```
mySet=set("abcd"); newSet=set("cdef")
```

```
mySet.symmetric_difference(newSet) returns set(['a','b','e','f'])
```

Michigan State University  
CSE 231, Fall 2009  
Sets and Dictionaries

### Set Ops, super and sub set

```
mySet=set("abc"); newSet=set("abcdef")
```

```
mySet.issubset(newSet) returns True
newSet.issuperset(mySet) returns True
```

Michigan State University  
CSE 231, Fall 2009  
Sets and Dictionaries

### Other Set Ops

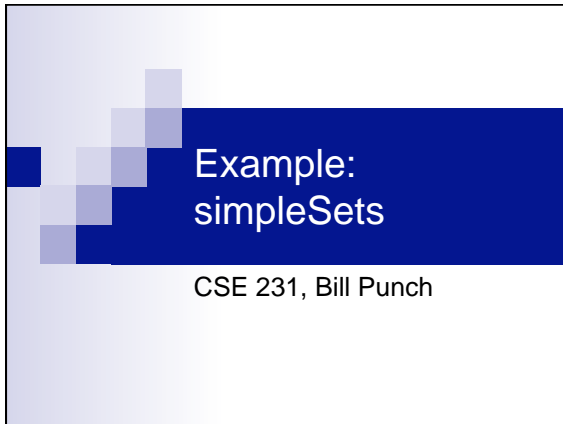
- `mySet.add("g")`
  - adds to the set, no effect if item is in set already
- `mySet.clear()`
  - empties the set
- `mySet.remove("g")` versus `mySet.discard("g")`
  - `remove` throws an error if "g" isn't there. `discard` doesn't care. Both remove "g" from the set
- `mySet.copy()`
  - returns a shallow copy of `mySet`

Michigan State University  
CSE 231, Fall 2009  
Sets and Dictionaries

### Copy vs. assignment

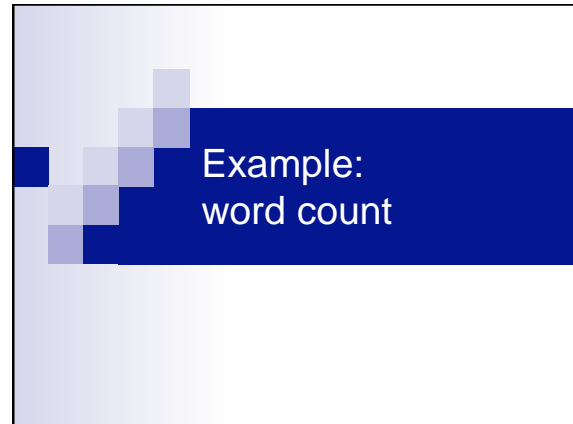
```
mySet=set("abc")
myCopy=mySet.copy()
myRefCopy=mySet
mySet.remove('b')
```

Michigan State University  
CSE 231, Fall 2009



Example:  
simpleSets

CSE 231, Bill Punch



Example:  
word count