Sets, Dictionaries and Inputs

Lecture 3

CSCI 3351 & CSCI 6651
Dr. Frank Breitinger

Overview

- Python doesn't have traditional vectors and arrays!
- Instead, Python makes heavy use of the dict datatype (a hashtable) which can serve as a sparse array
  - Efficient traditional arrays are available as modules that interface to C
- A Python set is derived from a dict

Sets

- Identified by curly braces
  - {'Alice', 'Bob', 'Carol'}
  - {'Dean'} is a singleton
- Can only contain unique elements
  - Duplicates are eliminated
- Immutable like tuples and strings
- Sets have no order and therefore do not support indexing.

Example

```
>>> alist = [11, 22, 33, 22, 44]
>>> aset = set(alist)
>>> aset
{33, 11, 44, 22}
>>> aset = aset + {55}
SyntaxError: invalid syntax
>>> aset[0]
Traceback (most recent call last):
   File "<pyshell#2>", line 1, in <module>
      myset[0]
TypeError: 'set' object does not support indexing
```

Boolean operations on sets

- `>>> aset = {11, 22, 33}`
- `>>> bset = {11, 44, 22, 55}`
- **Union** of two sets
  - `>>> aset | bset`
  - `>>> {33, 22, 55, 11, 44}`
- **Intersection** of two sets
  - `aset & bset`
  - `>>> {11, 22}`

Boolean operations on sets

- `>>> aset = {11, 22, 33}`
- `>>> bset = {11, 44, 22, 55}`
- **Difference** of two sets
  - `>>> aset - bset`
  - `>>> {33}`
  - `>>> bset - aset`
  - `>>> {44, 55}`
- **Symmetric** difference of two sets
  - `aset ^ bset`
  - `>>> {33, 44, 55}`

Utilization of sets

- Sets can be helpful to compare two lists for example:

```
>>> list1 = [11, "hello", 3.5]
>>> list2 = [11, "hello", 3.5]
>>> len(list1) == len(set(list1) & set(list2))
>>> True
```

- be careful!

```
list1.append("hello")
>>> print(list1)
>>> [11, 'hello', 3.5, 'hello']
prin1en(list1) == len(set(list1) & set(list2))
>>> False
len(list2) == len(set(list1) & set(list2))
>>> True
```

DICTIONARIES

Dictionaries: A Mapping type

- Dictionaries store a *mapping* between a set of keys and a set of values
  - Keys can be any *immutable* type.
  - Values can be any type
  - A single dictionary can store values of different types
- You can define, modify, view, lookup or delete the key-value pairs in the dictionary
- Python's dictionaries are also known as *hash tables* and *associative arrays*

Creating & accessing dictionaries

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
>>> d['bozo']
Traceback (innermost last):
  File '<interactive input>' line 1, in ?
KeyError: bozo
```

Updating Dictionaries

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user'] = 'clown'
>>> d
{'user':'clown', 'pswd':1234}
```

- Keys must be unique
- Assigning to an existing key replaces its value

```
>>> d['id'] = 45
>>> d
{'user':'clown', 'id':45, 'pswd':1234}
```

- Dictionaries are unordered
- New entries can appear anywhere in output
- Dictionaries work by hashing

Removing dictionary entries

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> del d['user']  # Remove one.
>>> d
{'p':1234, 'i':34}
>>> d.clear()      # Remove all.
>>> d
{}
>>> a=[1,2]
>>> del a[1]       # del works on lists, too
>>> a
[1]
```

Useful Accessor Methods

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}

>>> d.keys()  # List of keys, VERY useful
['user', 'p', 'i']

>>> d.values() # List of values
['bozo', 1234, 34]

>>> d.items()  # List of item tuples
[('user','bozo'), ('p',1234), ('i',34)]
```

```
>>> ages = { "Sam" : 4, "Mary" : 3, "Bill" : 2 }
>>> ages
{'Bill': 2, 'Mary': 3, 'Sam': 4}
>>> for name in ages.keys():
    print name, ages[name]
Bill 2
Mary 3
Sam 4
>>>
```

Summary

- Strings, lists, tuples, sets and dictionaries all deal with aggregates
- Two big differences
  - Lists and dictionaries are mutable
    - Unlike strings, tuples and sets
  - Strings, lists and tuples are ordered
    - Unlike sets and dictionaries

INPUTS

Overview

- There are three common ways to “communicate” with a program
  - user input
  - command line arguments
  - reading files
    - (not discussed in this lecture)
- Note, there is more but these are the most common ones, e.g.,
  - Network traffic, Interrupts, ...

User Input

- input: reads a string from the user's keyboard.
  - reads and returns an entire line of input

```
>>> name = input("Howdy. What's your name?")
Howdy. What's your name? Hans Wurst
>>> name
'Hans Wurst'
>>> name.lower()
```

User Input

- To read numbers, cast input result to an int or float
  - if the user does not type a number, an error occurs

```
age = int(input("How old are you? "))
print("Your age is", age)
print(65 - age, "years to retirement")
```

Some string methods

- len(**str**)
- startswith, endswith
- upper, lower,
- isupper, islower,
- capitalize, swapcase
- find
- strip

Command line arguments

- You can also give “arguments” to the program when you start it.
  - $ python scripts.py hello world
  - You call a program with 3 arguments, number one is always the program itself.

https://www.tutorialspoint.com/python/python_command_line_arguments.htm

Example

- Program code:

```
import sys
print(sys.argv[:])
```

- Call it on command line:

```
$ python scripts.py
hello world
```

- Output

```
['scripts.py',
'hello', 'world']
```

Question:

How do I get the amount of arguments passed to the program?

getopt.getopt method

- This method parses command line options and parameter list. Following is simple syntax for this method –
  - `getopt.getopt(args, options, [long_options])`
  - args: argument list to be parsed
  - options: string of option letters; options that allow argument are followed by a colon (:)
  - long_options: optional and ignored for today
- Example:
  - `usage: test.py -i <inputfile> -o <outputfile>`

Example

```
#!/usr/bin/python
import sys, getopt

def main(argv):
    opts, args = getopt.getopt(argv,"hi:o:")
    for opt, arg in opts:
        if opt == '-h':
            print('test.py -i <inputfile> -o <outputfile>')
            sys.exit()
        elif opt == "-i":
            inputfile = arg
            print('Input file is "', inputfile)
        elif opt == "-o":
            outputfile = arg
            print('Output file is "', outputfile)

main(sys.argv[1:])
```

Bonus assignment

- TBA