New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Advanced Python

Executive Summary, Session 2

STATEMENTS

- **print**: output string to STDOUT
- **ternary**: an if/else in a simple statement
- **conditional assignment**: if/else assignment

FUNCTIONS

- **sorted()**: return an iterable's items as a sorted list
- **bool()**: return **True** or **False** based on passed object
- **any()**: return True if any in an iterable is **True**
- **all()**: return True if all in an iterable are **True**

CORE OBJECT TYPES (continued)

DICTIONARY

```
mydict = { 'a': 1, 'b': 2, 'c': 3 }
```

• add a key/value pair	`mydict['d'] = 4`
• read a value based on a key	`var = mydict['d']`
• loop through a dict's keys	`for key in mydict:`
• loop through a dict's items	`for key, val in mydict.items()`
• check for key membership	`for key in mydict:`
• return length of a dict	`thislen = len(mydict)`
• **get()**: return a value (or default) given a key	`value = mydict.get('a', None)`
• **items()**: return a list of 2-element tuples	`items = mydict.items()`
• **keys()**: return a list of the dict's keys	`keys = mydict.keys()`
• **values()**: return a list of the dict's values	`values = mydict.values()`

SET

```
myset = { 'a', 'b', 'c', 'd' }    =or=     myset = set(['a', 'b', 'c', 'd'])
```

• **difference()**: return items in this set not in an iterable	`newset = myset.difference(this_list)`
• **union()**: return items in both this set and an iterable	`newset = myset.union(some_set)`
• **intersection()**: return items in this set also in an iterable	`newset = myset.intersection(a_tuple)`

FILE

```
fh = open('thisfile.txt')
```

• function: **open()** a file for writing	`fh = open('thisfile.txt', 'w')`
• function: **open()** a file for appending	`fh = open('thisfile.txt', 'a')`
• method: **write()**: write string data to the file	`fh.write('a line of text\n')`

MULTIDIMENSIONAL CONTAINERS

- list of lists

```
x = [  [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

  - access a single element

```
item = x[1][2]          # 6
```

  - loop through

```
for innerlist in x:
    for item in innerlist:
        print item
```

- list of dicts

```
x = [ { 'this': 5, 'that': 10 }, { 'this': 20, 'that': 7 } ]
```

  - access a single element

```
value = x[1]['that']      # 7
```

  - loop through

```
for innerdict in x:
    for key in innerdict:
        print innerdict[key]
```

- dict of lists

```
x = { 'a': [ 1, 2, 3, 4 ], 'b': [1, 3, 2, 4] }
```

    - access a single element

```
item = x['b'][1]              # 3
```

    - loop through

```
for key in x:
    for item in x[key]:
        print item
```

- dict of dicts

```
x = { 'a': { 'this': 5, 'that': 10 },  'b': { 'this': 20, 'that': 25 } }
```

    - access a single element

```
value = x['a']['that']          # 10
```

    - loop through

```
for key in x:
    print key + ':'
    for ikey in x[key]:
        print '    ' + ikey + ', ' + x[key][ikey]
```

LIST COMPREHENSIONS

```
wanted_lines = [line.split()[1] for line in lines if line.startswith('1972')]
```

USER-DEFINED FUNCTIONS

```
def addthese(arg1, arg2):
    mysum = arg1 + arg2
    return mysum
```

LAMBDAS

```
slist = sorted(mylist, key=lambda x: x.split()[0])
```

CUSTOM SORT FUNCTIONS

```
def by_first_item(line):
    items = line.split()
    first_item = items[0]
    return first_item
```

```
slist = sorted(mylist, key=by_first_item)
```

EXCEPTIONS

- **ValueError**: when the wrong value is used in a function, method or operation
- **KeyError**: when a key cannot be found in a **dict**
- **IndexError**: when an item index cannot be found in a **list**
- **OSError**: when the operating system signals an error to Python related to a file, directory or process

TRAPPING EXCEPTIONS

```
try:
      input = int(sys.argv[1])
   except (IndexError, ValueError):
       print input
```

MODULES

• **pprint**	`from pprint import pprint`
	`pprint(my_complex_struct)`

• **operator**	`import operator`
• **subprocess**	`import subprocess`
- **call()**	`subprocess.call(['ls', '-l'])`
- **check_output()**	`out_str = subprocess.check_output(['ls', '-l'])`

• multiprocessing.Process	`from multiprocessing import Process`
	`p = Process(target=workfunc, args=(myarg,))`

• os	`import os`
- listdir()	`files = os.listdir('some_directory')`
- path.isfile()	`if os.path.isfile('this_file'):`
- path.isdir()	`if os.path.isdir('this_dir'):`
- path.getsize()	`byteslen = os.path.getsize('this_file')`

COMMAND-LINE REDIRECTION (all snippets below at command line)

> STDOUT redirect to file	`./myprog.py > thisfile.txt`
| STDOUT redirect to another prog's STDIN	`./myprog.py > wc`
< STDIN redirect from file	`./reader.py < thatfile.txt`
| STDIN redirect from another prog's STDOUT	`ls -l > ./readdirlisting.py`