Programming Fundamentals and Python

Steven Bird Edward Loper Ewan Klein

University of Melbourne, AUSTRALIA

University of Pennsylvania, USA

University of Edinburgh, UK

July 9, 2006

Introduction

- non-technical overview
- many working program fragments
- try them for yourself as we go along
- many online tutorials (see `www.python.org`)
- Textbook: Zelle, John (2004) *Python Programming: An Introduction to Computer Science*

Introduction

- non-technical overview
- many working program fragments
- try them for yourself as we go along
- many online tutorials (see `www.python.org`)
- Textbook: Zelle, John (2004) *Python Programming: An Introduction to Computer Science*

Introduction

- non-technical overview
- many working program fragments
- try them for yourself as we go along
- many online tutorials (see `www.python.org`)
- Textbook: Zelle, John (2004) *Python Programming: An Introduction to Computer Science*

Introduction

- non-technical overview
- many working program fragments
- try them for yourself as we go along
- many online tutorials (see `www.python.org`)
- Textbook: Zelle, John (2004) *Python Programming: An Introduction to Computer Science*

Introduction

- non-technical overview
- many working program fragments
- try them for yourself as we go along
- many online tutorials (see `www.python.org`)
- Textbook: Zelle, John (2004) *Python Programming: An Introduction to Computer Science*

Defining Lists

- list: ordered sequence of items
- item: string, number, complex object (e.g. a list)
- list representation: comma separated items:
  `['John', 14, 'Sep', 1984]`
- list initialization:

```
>>> a = ['colourless', 'green', 'ideas']
```

- sets the value of variable `a`
- to see the its value, do: `print a`
- in interactive mode, just type the variable name:

```
>>> a
['colourless', 'green', 'ideas']
```

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple List Operations

1. length: `len()`
2. indexing: `a[0]`, `a[1]`
3. indexing from right: `a[-1]`
4. slices: `a[1:3]`, `a[-2:]`
5. concatenation: `b = a + ['sleep', 'furiously']`
6. sorting: `b.sort()`
7. reversing: `b.reverse()`
8. iteration: `for item in a:`
9. all the above applies to strings as well
10. double indexing: `b[2][1]`
11. finding index: `b.index('green')`

Simple String Operations

1. joining: `c = ' '.join(b)`
2. splitting: `c.split('r')`
3. lambda expressions: `lambda x:  len(x)`
4. maps: `map(lambda x:  len(x), b)`
5. list comprehensions: `[(x, len(x)) for x in b]`
6. getting help: `help(list),help(str)`

Simple String Operations

1. joining: `c = ' '.join(b)`
2. splitting: `c.split('r')`
3. lambda expressions: `lambda x: len(x)`
4. maps: `map(lambda x: len(x), b)`
5. list comprehensions: `[(x, len(x)) for x in b]`
6. getting help: `help(list), help(str)`

Simple String Operations

1. joining: `c = ' '.join(b)`
2. splitting: `c.split('r')`
3. lambda expressions: `lambda x:  len(x)`
4. maps: `map(lambda x:  len(x), b)`
5. list comprehensions: `[(x, len(x)) for x in b]`
6. getting help: `help(list),help(str)`

Simple String Operations

1. joining: `c = ' '.join(b)`
2. splitting: `c.split('r')`
3. lambda expressions: `lambda x:  len(x)`
4. maps: `map(lambda x:  len(x), b)`
5. list comprehensions: `[(x, len(x)) for x in b]`
6. getting help: `help(list), help(str)`

Simple String Operations

1. joining: `c = ' '.join(b)`
2. splitting: `c.split('r')`
3. lambda expressions: `lambda x:  len(x)`
4. maps: `map(lambda x:  len(x), b)`
5. list comprehensions: `[(x, len(x)) for x in b]`
6. getting help: `help(list), help(str)`

Simple String Operations

1. joining: `c = ' '.join(b)`
2. splitting: `c.split('r')`
3. lambda expressions: `lambda x:  len(x)`
4. maps: `map(lambda x:  len(x), b)`
5. list comprehensions: `[(x, len(x)) for x in b]`
6. getting help: `help(list)`, `help(str)`

Dictionaries

- accessing items by their names, e.g. dictionary
- defining entries:

```
>>> d = {}
>>> d['colourless'] = 'adj'
>>> d['furiously'] = 'adv'
>>> d['ideas'] = 'n'
```

- accessing:

```
>>> d.keys()
['furiously', 'colourless', 'ideas']
>>> d['ideas']
'n'
>>> d
{'furiously': 'adv', 'colourless': 'adj', 'ideas':
```

Dictionaries: Iteration

```
>>> for w in d:
...     print "%s [%s]," % (w, d[w]),
furiously [adv], colourless [adj], ideas [n],
```

- rule of thumb: dictionary entries are like variable names
- *create* them by assigning to them
  `x = 2` (variable), `d['x'] = 2` (dictionary entry)
- *access* them by reference
  `print x` (variable), `print d['x']` (dictionary entry)

Dictionaries: Example: Counting Word Occurrences

```
>>> from nltk_lite.corpora import gutenberg
>>> count = {}
>>> for word in gutenberg.raw('shakespeare-macbeth'):
...     word = word.lower()
...     if word not in count:
...         count[word] = 0
...     count[word] += 1
```

Now inspect the dictionary:

```
>>> print count['scotland']
12
>>> frequencies = [(freq, word) for (word, freq) in co
>>> frequencies.sort()
>>> frequencies.reverse()
>>> print frequencies[:20]
[(1986, ','), (1245, '.'), (692, 'the'), (654, "'"), (
```

Regular Expressions

- string matching
- substitution
- patterns, classes
- Python's regular expression module: `re`
- NLTK's utility function: `re_show`

Regular Expressions

- string matching
- substitution
- patterns, classes
- Python's regular expression module: `re`
- NLTK's utility function: `re_show`

Regular Expressions

- string matching
- substitution
- patterns, classes
- Python's regular expression module: `re`
- NLTK's utility function: `re_show`

Regular Expressions

- string matching
- substitution
- patterns, classes
- Python's regular expression module: `re`
- NLTK's utility function: `re_show`

Regular Expressions

- string matching
- substitution
- patterns, classes
- Python's regular expression module: `re`
- NLTK's utility function: `re_show`

Loading module, Matching

- Set up:

```
>>> import re
>>> from nltk_lite.utilities import re_show
>>> sent = "colourless green ideas sleep furio
```

- Matching:

```
>>> re_show('l', sent)
co{l}our{l}ess green ideas s{l}eep furious{l}y
>>> re_show('green', sent)
colourless {green} ideas sleep furiously
```

Substitutions

- E.g. replace all instances of `l` with `s`.
- Creates an output string (doesn't modify input)

```
>>> re.sub('l', 's', sent)
'cosoursess green ideas sseep furioussy'
```

- Work on substrings (NB not words)

```
>>> re.sub('green', 'red', sent)
'colourless red ideas sleep furiously'
```

More Complex Patterns

- Disjunction:

```
>>> re_show('(green|sleep)', sent)
colourless {green} ideas {sleep} furiously
>>> re.findall('(green|sleep)', sent)
['green', 'sleep']
```

- Character classes, e.g. non-vowels followed by vowels:

```
>>> re_show('[^aeiou][aeiou]', sent)
{co}{lo}ur{le}ss g{re}en{ i}{de}as s{le}ep {fu
>>> re.findall('[^aeiou][aeiou]', sent)
['co', 'lo', 'le', 're', ' i', 'de', 'le', 'fu
```

Structured Results

- Select a sub-part to be returned
- e.g. non-vowel characters which appear before a vowel:

```
>>> re.findall('([^aeiou])[aeiou]', sent)
['c', 'l', 'l', 'r', ' ', 'd', 'l', 'f', 'r']
```

- generate *tuples*, for later tabulation

```
>>> re.findall('([^aeiou])([aeiou])', sent)
[('c', 'o'), ('l', 'o'), ('l', 'e'), ('r', 'e'
```

Accessing Files and the Web

- accessing local files (create `corpus.txt` first)

```
>>> print open('corpus.txt').read()
Hello world.  This is a test file.
```

- Accessing URLs on the Web:

```
>>> from urllib import urlopen
>>> page = urlopen("http://news.bbc.co.uk/").r
>>> page = re.sub('<[^>]*>', '', page)
>>> page = re.sub('\s+', ' ', page)
>>> print page[:60]
 BBC NEWS | News Front Page News Sport Weather
```

Accessing NLTK

- modules: classes, functions
- data structures, algorithms
- importing, e.g. `from nltk_lite.utilities import *`
- conflicts
- explicit naming: `from nltk_lite.utilities import re_show`
- higher level module import:

```
>>> from nltk_lite import utilities
>>> utilities.re_show('green', s)
colourless {green} ideas sleep furiously
```

Texts from Project Gutenberg

```
>>> from nltk_lite.corpora import gutenberg
>>> gutenberg.items
['austen-emma', 'austen-persuasion', 'austen-sens
>>> count = 0
>>> for word in gutenberg.raw('whitman-leaves'):
...     count += 1
>>> print count
154873
```

Brown Corpus

```
>>> from nltk_lite.corpora import brown
>>> brown.items
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'l'
>>> from nltk_lite.corpora import extract
>>> print extract(0, brown.raw())
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', '
>>> print extract(0, brown.tagged())
[('The', 'at'), ('Fulton', 'np-tl'), ('County', 'nn-tl
```

Penn Treebank

```
>>> from nltk_lite.corpora import treebank
>>> print extract(0, treebank.parsed())
(S:
  (NP-SBJ:
    (NP: (NNP: 'Pierre') (NNP: 'Vinken'))
    (,: ',')
    (ADJP: (NP: (CD: '61') (NNS: 'years')) (JJ: 'old')
    (,: ',')))
  (VP:
    (MD: 'will')
    (VP:
      (VB: 'join')
      (NP: (DT: 'the') (NN: 'board'))
      (PP-CLR:
        (IN: 'as')
        (NP: (DT: 'a') (JJ: 'nonexecutive') (NN: 'dire
```