

# Programming Fundamentals and Python

Steven Bird   Ewan Klein   Edward Loper

University of Melbourne, AUSTRALIA

University of Edinburgh, UK

University of Pennsylvania, USA

August 27, 2008

# Introduction

- non-technical overview
- many working program fragments
- try them for yourself as we go along
- many online tutorials (see [www.python.org](http://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](http://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](http://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](http://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](http://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 its value, do: print a
- in interactive mode, just type the variable name:

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

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0], a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3], a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0], a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3], a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple List Operations

- ➊ length: `len()`
- ➋ indexing: `a[0]`, `a[1]`
- ➌ indexing from right: `a[-1]`
- ➍ slices: `a[1:3]`, `a[-2:]`
- ➎ concatenation: `b = a + ['sleep', 'furiously']`
- ➏ sorting: `b.sort()`
- ➐ reversing: `b.reverse()`
- ➑ iteration: `for item in a:`
- ➒ all the above applies to strings as well
- ➓ double indexing: `b[2][1]`
- ➔ finding index: `b.index('green')`

# Simple String Operations

- ➊ joining: `c = ' '.join(b)`
- ➋ splitting: `c.split('r')`
- ➌ lambda expressions: `lambda x: len(x)`
- ➍ maps: `map(lambda x: len(x), b)`
- ➎ list comprehensions: `[(x, len(x)) for x in b]`
- ➏ getting help: `help(list), help(str)`

# Simple String Operations

- ➊ joining: `c = ' '.join(b)`
- ➋ splitting: `c.split('r')`
- ➌ lambda expressions: `lambda x: len(x)`
- ➍ maps: `map(lambda x: len(x), b)`
- ➎ list comprehensions: `[(x, len(x)) for x in b]`
- ➏ getting help: `help(list), help(str)`

# Simple String Operations

- ➊ joining: `c = ' '.join(b)`
- ➋ splitting: `c.split('r')`
- ➌ lambda expressions: `lambda x: len(x)`
- ➍ maps: `map(lambda x: len(x), b)`
- ➎ list comprehensions: `[(x, len(x)) for x in b]`
- ➏ getting help: `help(list), help(str)`

# Simple String Operations

- ➊ joining: `c = ' '.join(b)`
- ➋ splitting: `c.split('r')`
- ➌ lambda expressions: `lambda x: len(x)`
- ➍ maps: `map(lambda x: len(x), b)`
- ➎ list comprehensions: `[(x, len(x)) for x in b]`
- ➏ 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

```
>>> import nltk
>>> count = {}
>>> for word in nltk.corpus.gutenberg.words('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 count.items()]
>>> frequencies.sort()
>>> frequencies.reverse()
>>> print frequencies[:20]
[(1986, ','), (1245, '.'), (692, 'the'), (654, ""), (567, 'and'), (566, 'to'), (550, 'a'), (540, 'of'), (530, 'in'), (520, 'is'), (510, 'that'), (500, 'not'), (490, 'he'), (480, 'you'), (470, 'with'), (460, 'and'), (450, 'as'), (440, 'on'), (430, 'but'), (420, 'for'), (410, 'are')]
```

# 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 nltk, re  
>>> sent = "colourless green ideas sleep furiously"
```

- Matching:

```
>>> nltk.re_show('l', sent)  
co{l}our{l}ess green ideas s{l}eep furious{l}y  
>>> nltk.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 furiouussy'
```

- Work on substrings (NB not words)

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

# More Complex Patterns

- Disjunction:

```
>>> nltk.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:

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

# 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'), (' ', 'e'), ('d', 'e'), ('f', '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/").read()
>>> text = nltk.clean_html(page)
>>> print text[:60]
BBC NEWS | News Front Page News Sport Weather Wor...
```

# Accessing NLTK

- modules: classes, functions
- data structures, algorithms
- importing, e.g. `import nltk`

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

# Texts from Project Gutenberg

```
>>> nltk.corpus.gutenberg.items  
['austen-emma', 'austen-persuasion', 'austen-sense', ''  
>>> count = 0  
>>> for word in nltk.corpus.gutenberg.words('whitman-1'  
...     count += 1  
>>> print count  
154873
```

# Brown Corpus

```
>>> nltk.corpus.brown.items  
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'n', 'p',  
>>> print nltk.corpus.brown.words('a')  
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an',  
>>> print nltk.corpus.brown.tagged_sents('a')  
[('The', 'at'), ('Fulton', 'np-tl'), ('County', 'nn-tl'), ('Grand',
```

# Penn Treebank

```
>>> print nltk.corpus.treebank.parsed_sents('wsj_0001')[0]
(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: 'director')))
      (NP-TMP: (NNP: 'Nov.') (CD: '29'))))
    (..: '..')))
```