SciComp with Py

List Comprehension

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- Review
- Set Former Notation
- List Comprehension

4 Main Iterable Types

- There are 4 main iterables in Py (data structures that can be iterated over element by element):

    1) Strings
    2) Lists
    3) Tuples
    4) Sets

Strings

- Strings are **built-in**, **homogeneous**, **indexed**, & **immutable** sequences of characters enclosed in matching double or single quotes
- Examples: **“one”, 'two', “abracadabra”, 'abracadabra'**
- String indexing is **0**-based
- Strings are immutable: cannot be assigned into, only read out of

Lists

- Lists are **built-in**, **heterogeneous**, **indexed**, & **mutable** sequences
- A list is any sequence of valid Py elements separated by commas and enclosed in **[ ]**
- Lists are heterogeneous: **[1, 2, 3], [1, 'one', 2, “two”]**
- List indexing is **0**-based, i.e., indexes start at **0**
- A list can be assigned to a regular variable (so long as the variable's name is legal)
- Examples:

```
lst0 = [1, 2, 3]
lst1 = [1, 'one', 2, 'two']
```

Tuples

- Tuples are **built-in**, **heterogeneous**, **indexed**, & **immutable** sequences of elements enclosed in **( )**
- Examples: **(1, 2, 3)**, **(1, “one”, 2, 'two')**
- Indexing is **0**-based
- Tuples are immutable: cannot be assigned into, only read out of

Sets

- Sets are **built-in**, **heterogeneous**, **unindexed**, & **mutable**
- Sets can contain numbers, strings, and tuples; they cannot contain other lists or sets
- You can iterate over sets, add and remove elements

Construction of Sets

Py2

```
>>> s1 = set([1, 2, 3])
>>> s2 = set(('a', 5.5, (10, 11)))
>>> s1
set([1, 2, 3])
>>> s2
set(['a', 5.5, (10, 11)])
```

Py3

```
>>> s1 = set([1, 2, 3])
>>> s2 = set(('a', 5.5, (10, 11)))
>>> s1
{1, 2, 3}
>>> s2
{(10, 11), 5.5, 'a'}
```

Checking Set Membership

Py2

```
>>> 1 in s1
True
>>> 'a' in s2
True
>>> (10, 11) in s2
True
>>> 'b' in s2
False
```

Py3

```
>>> 1 in s1
True
>>> 'a' in s2
True
>>> (10, 11) in s2
True
>>> 'b' in s2
False
```

Adding/Removing Elements To/From Sets

Py2

```
>>> s1
set([1, 2, 3])
>>> s1.add(5)
>>> s1
set([1, 2, 3, 5])
>>> s1.remove(1)
>>> s1
set([2, 3, 5])
```

Py3

```
>>> s1
{1, 2, 3}
>>> s1.add(5)
>>> s1
{1, 2, 3, 5}
>>> s1.remove(1)
>>> s1
{2, 3, 5}
```

Sequence Slicing

- If **seq** is a sequence (list, string, toop), a **slice** of that **seq** is a sub-sequence of elements
- For example, if **seq** is a list, then a slice is a sub-list; if **seq** is a tuple, then a slice is a tuple; if **seq** is a string, a slide is a sub-string
- A slice is specified by its start and end indexes; e.g., **seq[start:end]**
- A slice **seq[start:end]** includes all elements from **seq[start]** upto **seq[end-1]**

Sequence Slicing

Both negative and non-negative indices can be used in slicing. Slicing tuples and strings can be done in the same way. Examples:

```
    lst = ['one', 'two', 'three', 'four', 'five']
    slice_01 = lst[1:4] ## ['two', 'three', 'four']
    lst = [1, 2, 3, 4, 5]
    lst[-4:-1] == [2, 3, 4] ## returns True
 ## lst[-4:-1] consists of lst[-4], lst[-3], and lst[-2]. The
 ## element lst[-1] is not included.
```

A Few Other Things to Review in Lecture 2

- Make sure you understand numerical ranges **xrange()** and **range()**
- Get comfortable with anonymous functions, aka lambdas, and know the difference b/w anonymous and named functions
- Go through the sequence iteration examples with for-loops

Set Former Notation

List Comprehension & Set-Former Notation

- **List comprehension** is an syntactic construct in some programming languages for building lists from their specifications
- **List comprehension** derives its conceptual roots from the ***set-former*** (***set-builder***) notation in mathematics
- **List comprehension** is available in other programming languages such as Lisp, Scheme, Haskell, and Ocaml

Set-Former Notation

$\{2\cdot x \,||\, x\in N, 2\,|\,x, x<11\} = \{2\cdot 0, 2\cdot 2, 2\cdot 4, 2\cdot 6, 2\cdot 8, 2\cdot 10\} =$
$\{0,4,8,12,16,20\}$; the symbol $||$ typicaly reads "such as."

$\{2\cdot x \,||\, x\in N, 2\,|\,x, x<11\}$

– x is the input variable

– $2\cdot x$ is the output function

– N is the input set; the domain of the output function

– $2\,|\,x, x<11$ is the predicate

Set Former Notation

$$\{4x \| x \in N, x^2 < 100\} = \{4\cdot 0, 4\cdot 1, 4\cdot 2, 4\cdot 3, 4\cdot 4, 4\cdot 5, 4\cdot 6, 4\cdot 7, 4\cdot 8, 4\cdot 9\} =$$
$$\{0,4,8,12,16,20,24,28,32,36\}$$

$$\{4\cdot x \| x \in N, x^2 < 100\}$$

– $4\cdot x$ is the output function

– x is the input variable

– N is the input set; the domain of the output function

– $x^2 < 100$ is the predicate

Construction of Lists
with
List Comprehension

Problem 1

Write a Py program that builds $\left\{2\cdot x \,||\, x\in N, 2\,|\,x, x<11\right\}$ with a loop.

Write a Py program that builds $\left\{2\cdot x \,||\, x\in N, 2\,|\,x, x<11\right\}$ with list comprehension.

source code is in py2_list_comp.py & py3_list_comp.py

Solution 1a: Loops

Build a List

```
def build_list_1a():
    rslt = []
    x = 0
    while x < 11:
        if x % 2 == 0:
            rslt.append(2*x)
        x += 1
    return rslt

print build_list_1a()
```

Py Output

```
[0, 4, 8, 12, 16, 20]
```

Build a Set

```
def build_set_1a():
    rslt = set()
    x = 0
    while x < 11:
        if x % 2 == 0:
            rslt.add(2*x)
        x += 1
    return rslt

print build_set_1a()
```

Py Output

```
set([0, 4, 8, 12, 16, 20])
```

Solution 1b: List Comprehension

Building [0, 4, 8, 12, 16, 20]

```
def build_list_1b(): return [2*x for x in xrange(11) if x % 2 == 0]
```

Building {0, 4, 8, 12, 16, 20}

```
def build_set_1b(): return set([2*x for x in xrange(11) if x % 2 == 0])
```

Solutions 1a & 1b Side by Side

1a

```
def build_list_1a():
    rslt = []
    x = 0
    while x < 11:
        if x % 2 == 0:
            rslt.append(2*x)
        x += 1
    return rslt
```

1b

```
def build_list_1b():
    return [2*x for x in xrange(11) if x % 2 == 0]
```

Note the difference in the amount of code b/w the two solutions

Problem 2

Write a Py program that builds $\{4x \,||\, x \in N, x^2 < 100\}$ with a loop.

Write a Py program that builds $\{4x \,||\, x \in N, x^2 < 100\}$ with list comprehension.

source code is in py2_list_comp.py & py3_list_comp.py

For the sake of brevity, I will skip building sets and focus on lists from now on; I will also focus on Py2; Py3 solutions are very similar – remember to change **xrange** to **range** and add parentheses after **print**

Solution 2a: Loop

Py

```
  def build_list_2a():
      rslt = []
      x = 0
      while x**2 < 100:
          rslt.append(4*x)
          x += 1
      return rslt

print build_list_2a()
```

Py Output

```
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36]
```

Solution 2b: List Comprehension

Py

```
def build_list_2b(): return [4*x for x in xrange(101) if x**2 < 100]

print build_list_2b()
```

Py Output

```
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36]
```

Solutions 2a & 2b Side by Side

2a

```
def build_list_2a():
    rslt = []
    x = 0
    while x**2 < 100:
        rslt.append(4*x)
        x += 1
    return rslt
```

2b

```
def build_list_2b():
    return [4*x for x in xrange(101) if x**2 < 100]
```

Note the difference in the amount of code b/w 2a and 2b

Problem 3

Write a Py program that builds $\left\{x^3 \,\|\, x \in N, \neg\left(2 \mid x\right), x < 11\right\}$ with a loop.

Write a Py program that builds $\left\{x^3 \,\|\, x \in N, \neg\left(2 \mid x\right), x < 11\right\}$ with list comprehension.

source code is in py2_list_comp.py & py3_list_comp.py

Solution 3a: Loop

Py

```
def build_list_3a():
    rslt = []
    x = 0
    while x < 11:
        if x % 2 != 0:
            rslt.append(x**3)
        x += 1
    return rslt

print build_list_3a()
```

Py Output

```
[1, 27, 125, 343, 729]
```

Solution 3b: List Comprehension

Py

```
def build_list_3b(): return [x**3 for x in xrange(11) if x % 2 != 0]

print build_list_3b()
```

Py Output

```
[1, 27, 125, 343, 729]
```

Solutions 3a & 3b Side by Side

3a

```
def build_list_3a():
    rslt = []
    x = 0
    while x < 11:
        if x % 2 != 0:
            rslt.append(x**3)
        x += 1
    return rslt
```

3b

```
def build_list_3b():
    return [x**3 for x in xrange(11) if x % 2 != 0]
```

Note the difference in the amount of code b/w the two solutions

Problem 4

Write a Py program that builds

$$\left\{\left(x, y\right) \| x \in N, y \in N, \left(2 \mid x\right), \neg\left(2 \mid y\right), x \leq 5, y \leq 5\right\}$$

with a loop.

Write a Py program that builds

$$\left\{\left(x, y\right) \| x \in N, y \in N, \left(2 \mid x\right), \neg\left(2 \mid y\right), x \leq 5, y \leq 5\right\}$$

with list comprehension.

source code is in py2_list_comp.py & py3_list_comp.py

Solution 4a: Loops

Py

```
def build_list_4a():
    rslt = []
    for x in xrange(6):
        if x % 2 == 0:
            for y in xrange(6):
                if not y % 2 == 0:
                    rslt.append((x,y))
    return rslt

print build_list_4a()
```

Py Output

```
[(0, 1), (0, 3), (0, 5), (2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)]
```

Solution 4b: List Comprehension

Py

```
def build_list_4b():
    return [(x,y)
            for x in xrange(6) if x % 2 == 0
            for y in xrange(6) if not y % 2 == 0]


print build_list_4b()
```

Py Output

```
[(0, 1), (0, 3), (0, 5), (2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)]
```

Solutions 4a & 4b Side by Side

4a

```
def build_list_4a():
    rslt = []
    for x in xrange(6):
        if x % 2 == 0:
            for y in xrange(6):
                if not y % 2 == 0:
                    rslt.append((x,y))
    return rslt
```

4b

```
def build_list_4b():
    return [(x,y)
            for x in xrange(6) if x % 2 == 0
            for y in xrange(6) if not y % 2 == 0]
```

Note the difference in the amount of code b/w the two solutions

Problem

Write a Py program that computes the sum of all numbers in a 2D matrix with a loop.

Write a Py program that computes the sum of all numbers in a 2D matrix with list comprehension.

source code is in py2_mats.py & py3_mats.py

Sample Matrices as Lists & Tuples

```
mat = \
[
    [1, 1, 1],
    [2, 2, 2],
    [3, 3, 3]
]

toop_mat = \
(
    (1, 1, 1),
    (2, 2, 2),
    (3, 3, 3)
)
```

```
mat2 = \
[
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8]
]

toop_mat2 = \
(
    (0, 1, 2),
    (3, 4, 5),
    (6, 7, 8)
)
```

Solution with Loops

Py

```
def mat_loop_sum(mat):
    sum_total = 0
    for r in mat:
        sum_total += sum(r)
    return sum_total

print 'list matrix sum w/ loops =', mat_loop_sum(mat)
print 'toop matrix sum w/ loops = ', mat_loop_sum(toop_mat)
```

Py Output

```
list matrix sum w/ loops = 18
toop matrix sum w/ loops =  18
```

Solution with List Comprehension

Py

```
def mat_listcomp_sum(mat): return sum([sum(r) for r in mat])

print 'list matrix sum w/ listcomp = ', mat_listcomp_sum(mat)
print 'toop matrix sum w/ listcomp = ', mat_listcomp_sum(mat)
```

Py Output

```
list matrix sum w/ listcomp =  18
toop matrix sum w/ listcomp =  18
```

Problem

Write a Py program that computes the sum of all numbers in a specific column in a 2D matrix with a loop.

Write a Py program that computes the sum of all numbers in a specific column in a 2D matrix with list comprehension.

Solution w/ Loops

Py

```
def display_mat(mat):
    for row in mat:
        print row
    print

def mat_loop_col_sum(mat, col_num):
    col_sum_total = 0
    for r in mat:
        col_sum_total += r[col_num]
    return col_sum_total

def test_mat_loop_col_sum(mat, num_cols):
    print 'list matrix column sums w/ loops for matrix:'
    display_mat(mat)
    for col_num in xrange(num_cols):
        print 'sum of column', str(col_num), '=', mat_loop_col_sum(mat, col_num)
    print

test_mat_loop_col_sum(mat2, 3)
```

Py Output

```
list matrix column sums w/ loops for matrix:
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]

sum of column 0 = 9
sum of column 1 = 12
sum of column 2 = 15
```

Solution w/ List Comprehension

Py

```
def display_mat(mat):
    for row in mat:
        print row
    print

def mat_listcomp_col_sum(mat, col_num):
    return sum([row[col_num] for row in mat])

def test_mat_listcomp_col_sum(mat, num_cols):
    print 'list matrix column sums w/ list comprehension for matrix:'
    display_mat(mat)
    for col_num in xrange(num_cols):
        print 'sum of column', str(col_num), '=', mat_listcomp_col_sum(mat, col_num)
    print

test_mat_listcomp_col_sum(toop_mat2, 3)
```

Py Output

```
list matrix column sums w/ list comprehension for matrix:
(0, 1, 2)
(3, 4, 5)
(6, 7, 8)

sum of column 0 = 9
sum of column 1 = 12
sum of column 2 = 15
```