Higher-Order Functions; List Comprehensions

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

Another Useful Built-In Function

- `sum(list)`: computes & returns the sum of a list of numbers

```
>>> sum([4, 10, 2])
16
```

Useful Built-In Functions (cont.)

- `range(low, high)`: allows us to work with the range of integers from `low` to `high-1`
    - if you omit `low`, the range will start at 0
- You can think of `range` as producing a list, and in many cases it can be used like one.
- To *see* the actual list, we need to use `range` in conjunction with another function called `list`:

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(7))
[0, 1, 2, 3, 4, 5, 6]
```

map()

- A higher-order function
- Syntax:

    `map(`*function*, *sequence*`)`

    - applies *function* to each element of *sequence* and returns the results
- As with `range`:
    - you can think of `map` as producing a list
    - in many cases it can be used like one
    - to *see* the actual list, we need to use `map` with `list`

map() Examples

```
def triple(x):
    return 3*x

def square(x):
    return x*x

def first_char(s):
    return s[0]
```

```
>>> list(map(triple, [0, 1, 2, 3, 4, 5]))
[0, 3, 6, 9, 12, 15]
                                [0, 1, 2, 3, 4, 5]
>>> list(map(square, range(6)))
[0, 1, 4, 9, 16, 25]

>>> list(map(first_char, ['python', 'is', 'fun!']))
['p', 'i', 'f']

>>> list(map(triple, 'python'))
['ppp', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
```

List Comprehensions

The same as `map`, *only better!*

[*expression* `for` *variable* `in` *sequence*]

```
>>> [3*x for x in [0,1,2,3,4,5]]
```

List Comprehensions

The same as `map`, *only better!*

`[`*expression* `for` *variable* `in` *sequence*`]`

```
>>> [3*x for x in [0,1,2,3,4,5]]
```

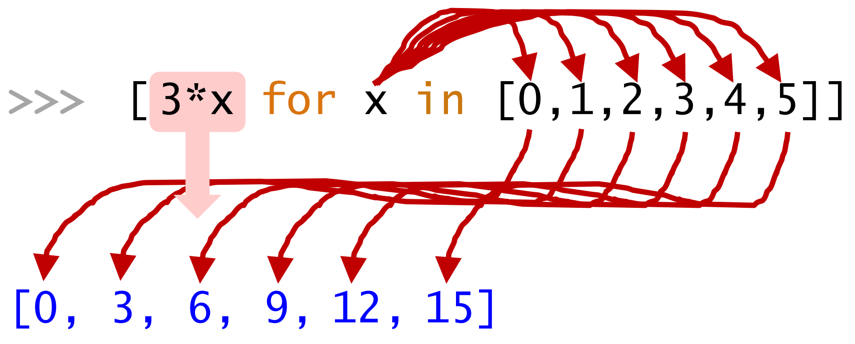
x takes on each value

and **3*x** is output for each one

List Comprehensions

The same as `map`, *only better!*

`[`*expression* `for` *variable* `in` *sequence*`]`

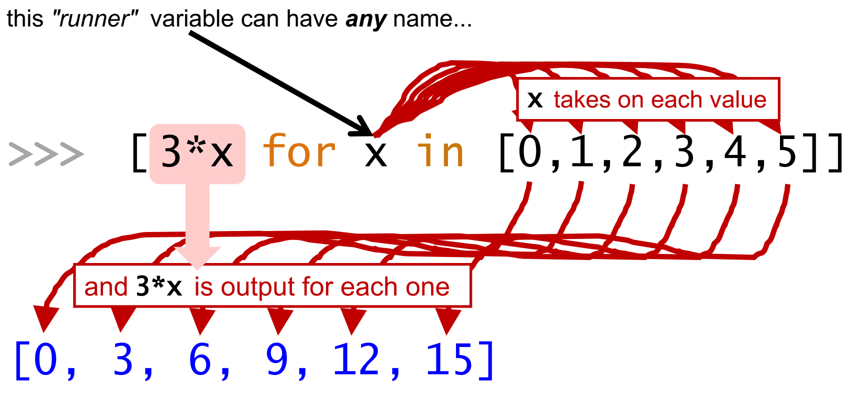


```
>>> [3*x for x in [0,1,2,3,4,5]]
[0, 3, 6, 9, 12, 15]
```

List Comprehensions

The same as `map`, *only better!*

```
[expression for variable in sequence]
```


this *"runner"* variable can have ***any*** name...
x takes on each value
and **3*x** is output for each one

```
>>> [3*x for x in [0,1,2,3,4,5]]
[0, 3, 6, 9, 12, 15]
```

List Comprehensions (LCs)

The same as `map`, *only better!*

- Syntax:

```
[expression for variable in sequence]
                 or
[expression for variable in sequence  if boolean]
```

More Examples

```
               [10, 11, 12, 13, 14]
>>> [n - 2 for n in range(10, 15)]
[8, 9, 10, 11, 12]

>>> [s[-1]*2 for s in ['go', 'terriers!']]
['oo', '!!']

>>> [z for z in range(6)]
[0, 1, 2, 3, 4, 5]

>>> [z for z in range(6) if z % 2 == 1]
[1, 3, 5]

>>> [z % 4 == 0 for z in [4, 5, 6, 7, 8]]
[True, False, False, False, True]

>>> [1 for x in [4, 5, 6, 7, 8] if x % 4 == 0]
[1, 1]

>>> sum([1 for x in [4, 5, 6, 7, 8] if x % 4 == 0])
2
```

What is the output of this code?

```
lc = [x for x in range(5) if x**2 > 4]

print(lc)
```

A. `[9, 16]`
B. `[9, 16, 25]`
C. `[3, 4]`
D. `[3, 4, 5]`
E. none of these

What is the output of this code?

```
lc = [x for x in range(5) if x**2 > 4]

print(lc)
```

A. `[9, 16]`

B. `[9, 16, 25]`

C. **`[3, 4]`**

D. `[3, 4, 5]`

E. none of these

What is the output of this code?

```
lc = [x for x in range(5) if x**2 > 4]
                 [0,1,2,3,4]
print(lc)        [0,1,4,9,16]  ← x**2
```

A. `[9, 16]`

B. `[9, 16, 25]`

C. **`[3, 4]`**

D. `[3, 4, 5]`

E. none of these

LC Puzzles! – Fill in the blanks

```
>>> [__________ for x in range(4)]
[0, 14, 28, 42]

>>> [__________ for s in ['boston', 'university', 'cs']
['bos', 'uni', 'cs']

>>> [__________ for c in 'compsci']
['cc', 'oo', 'mm', 'pp', 'ss', 'cc', 'ii']

>>> [__________ for x in range(20, 30) if ____________]
[20, 22, 24, 26, 28]

>>> [__________ for w in ['I', 'like', 'ice', 'cream']]
[1, 4, 3, 5]
```

LC Puzzles! – Fill in the blanks

```
>>> [   14*x    for x in range(4)]
[0, 14, 28, 42]

>>> [   s[:3]   for s in ['boston', 'university', 'cs']
['bos', 'uni', 'cs']

>>> [    c*2    for c in 'compsci']
['cc', 'oo', 'mm', 'pp', 'ss', 'cc', 'ii']

>>> [     x     for x in range(20, 30) if  x % 2 == 0 ]
[20, 22, 24, 26, 28]

>>> [  len(w)   for w in ['I', 'like', 'ice', 'cream']]
[1, 4, 3, 5]
```

LCs vs. Raw Recursion

```
# raw recursion
def mylen(seq):
   if seq == '' or seq == []:
       return 0
   else:
       len_rest = mylen(seq[1:])
       return 1 + len_rest

# using an LC
def mylen(seq):
   lc = [1 for x in seq]
   return sum(lc)

# here's a one-liner!
def mylen(seq):
   return sum([1 for x in seq])
```

LCs vs. Raw Recursion (cont.)

```
# raw recursion
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_in_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1 + num_in_rest
        else:
            return 0 + num_in_rest

# using an LC
def num_vowels(s):
   lc = [1 for c in s if c in 'aeiou']
   return sum(lc)

# here's a one-liner!
def num_vowels(s):
   return sum([1 for c in s if c in 'aeiou'])
```

What list comprehension(s) would work here?

```
def num_odds(values):
    """ returns the number of odd #s in a list
        input: a list of 0 or more integers
    """
    lc = ____________________________________
    return sum(lc)
```

A. `[x for x in values if x // 2 == 1]`

B. `[1 for x in values if x // 2 == 1]`

C. `[x for x in values if x % 2 == 1]`

D. `[1 for x in values if x % 2 == 1]`

E. none of these

What list comprehension(s) would work here?

```
def num_odds(values):
    """ returns the number of odd #s in a list
        input: a list of 0 or more integers
    """
    lc = ____________________________________
    return sum(lc)
```

A. `[x for x in values if x // 2 == 1]`

B. `[1 for x in values if x // 2 == 1]`

C. `[x for x in values if x % 2 == 1]`

D. **`[1 for x in values if x % 2 == 1]`**

E. none of these

Fill in the Blanks

```
def avg_len(wordlist):
    """ returns the average length of the strings
        in wordlist as a float
        input: a list of 1 or more strings
    """
    lc = [_________ for ____ in __________]

    return ________ / _________



>>> avg_len(['commonwealth', 'avenue'])
9.0
>>> avg_len(['keep','calm','and','code','on'])
3.4
```

Fill in the Blanks

```
def avg_len(wordlist):
    """ returns the average length of the strings
        in wordlist as a float
        input: a list of 1 or more strings
    """
    lc = [len(word) for word in wordlist]

    return sum(lc) / len(lc)  # or len(wordlist)



>>> avg_len(['commonwealth', 'avenue'])
9.0
>>> avg_len(['keep','calm','and','code','on'])
3.4
```

What is the output of this program?

```
def myst(s):
    lc = [c for c in s if c != 'a']
    return lc

result = myst('banana')
print(result)
```

A. `['banana']`

B. `['bnn']`

C. `['b n n ']`

D. `['b', 'n', 'n']`

E. `['b', '', 'n', '', 'n', '']`

What is the output of this program?

```
def myst(s):
    lc = [c for c in s if c != 'a']
    return lc

result = myst('banana')
print(result)
```

s = 'banana'

c	c != 'a'	lc
'b'	True	['b']
'a'	False	['b']
'n'	True	['b', 'n']
'a'	False	['b', 'n']
'n'	True	['b', 'n', 'n']
'a'	False	['b', 'n', 'n']

A. `['banana']`

B. `['bnn']`

C. `['b n n ']`

D. **`['b', 'n', 'n']`**

E. `['b', '', 'n', '', 'n', '']`