



COMPSCI 105 S1 2017 Principles of Computer Science

Lists & List comprehensions



Exercises

- ▶ What is the output of the following code fragment?

```
def ex6():
    i = 0
    while i < 5:
        print(i)
        i += 1
        if i == 3:
            break
    else:
        print(0)
ex6()
```

```
print (True or False and False)
print( (True or False) and False )
```

```
value = 12
print( value>10 or value<=5 and value!=12)
print( (value>10 or value<=5) and value!=12)
```



Lists

- ▶ Lists are a built-in type in Python
 - ▶ Use square brackets to signify a list
 - ▶ Lists can contain any type of data, or any mixture of data

```
my_list1 = [1, 2, 3]
my_list2 = ['Hello', 'Is', 'there', 'anybody', 'out', 'there?']
my_list3 = [1, 5.899, 'Hello']
my_list4 = [4, 2, 6, 9, 3]
```



List functions

- ▶ Numerous list functions are supported

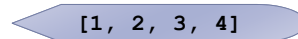
- ▶ Use help(list) to find out the functions

- ▶ Examples:

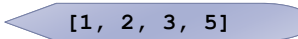
```
>>> x = [1, 2, 3]
>>> len(x)
```



```
>>> x + [4]
```



```
>>> x += [5]
```



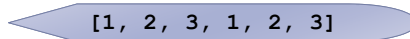
```
>>> 3 in x
```



```
>>> x[0]
```



```
>>> [1, 2, 3] * 2
```





List comprehensions

- ▶ A powerful feature of the Python language.
 - ▶ A list can be created using instructions that appear within the square brackets
 - ▶ Generate a new list by applying a function to every member of an original list.
- ▶ The syntax of a “list comprehension” is tricky.
 - ▶ If you’re not careful, you might think it is a for-loop, an ‘in’ operation, or an ‘if’ statement since all three of these keywords (‘for’, ‘in’, and ‘if’) can also be used in the syntax of a list comprehension.
 - ▶ It’s something special all its own.

```
my_list = [x for x in range(0, 10)]
```

5

COMPSCI 105

Lecture 02



List comprehensions : Syntax 1

- ▶ The general format is as follows:
 - ▶ Where expression is some calculation or operation acting upon the variable.
 - ▶ For each member of the sequence, calculate a new value using expression, and then we collect these new values into a new list which becomes the return value of the list comprehension.
- ▶ Examples:

```
my_list = [c for c in 'Ann']
```

```
['A', 'n', 'n']
```

```
freshfruit = [' banana', ' loganberry ']
li = [weapon.strip() for weapon in freshfruit]
```

```
['banana', 'loganberry']
```

6

COMPSCI 105

Lecture 02



Some more examples...

- ▶ Calculation...

```
vec = [2, 4, 6]
li = [3*x for x in vec]
```

```
[6, 12, 18]
```

- ▶ Using a range function:

```
li = [n * 2 for n in range(1, 5)]
```

```
[2, 4, 6, 8]
```

7

COMPSCI 105

Lecture 02



Exercise 1

- ▶ Write a list comprehension that generates all the odd numbers between 1 and 50

8

COMPSCI 105

Lecture 02



List Comprehension Syntax 2

- ▶ If the original list contains a variety of different types of values, then the calculations contained in the expression should be able to operate correctly on all of the types of list members.

```
values = ['hello', [1,2], (3,5)]
li = [len(n) for n in values]
```

[5, 2, 2]

- ▶ If the members of list are other containers, then the name can consist of a container of names that match the **type** and **“shape”** of the list members.

```
values = [('a', 1), ('b', 2), ('c', 7)]
li = [n * 3 for (x, n) in values]
```

[3, 6, 21]

9

COMPSCI 105

Lecture 02



List Comprehension Syntax 3

- ▶ The expression of a list comprehension could also contain user-defined functions.

```
def foo(a):
    return a * a
```

```
oplist = [(6, 3), (1, 7), (5, 5)]
li = [foo(x) for (x, y) in oplist]
```

[36, 1, 25]

- ▶ Example 2:

```
def subtract(a, b):
    return a - b
```

[-3, 6, 0]

```
oplist = [(6, 3), (1, 7), (5, 5)]
li = [subtract(y, x) for (x, y) in oplist]
```

10

COMPSCI 105

Lecture 02



List Comprehension Syntax 4

- ▶ We can also create ...

- ▶ a list of dictionaries:

[{2: 4}, {4: 16}, {6: 36}]

```
vec = [2, 4, 6]
li = [{x: x**2} for x in vec]
```

- ▶ a list of list:

[[2, 4], [4, 16], [6, 36]]

```
vec = [2, 4, 6]
li = [[x, x**2] for x in vec]
```

11

COMPSCI 105

Lecture 02



List Comprehension Syntax 5

- ▶ With Two sequences...

```
vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
li = [x*y for x in vec1 for y in vec2]
```

[8, 6, -18, 16, 12, -36, 24, 18, -54]

- ▶ $2*4, 2*3, 2*(-9)$
- ▶ $4*4, 4*3, 4*(-9)$
- ▶ $6*4, 6*3, 6*(-9)$

```
li = [x+y for x in vec1 for y in vec2]
```

[6, 5, -7, 8, 7, -5, 10, 9, -3]

12

COMPSCI 105

Lecture 02



List Comprehension Syntax 5

- ▶ With Two sequences...

```
vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
li = [vec1[i]*vec2[i] for i in range(len(vec1))]
```

- ▶ $i=0, i=1, i=2$ (execute three times)
- ▶ $2*4, 4*3, 6*(-9)$

[8, 12, -54]



List comprehensions that use conditions (Filtered List)

- ▶ We can extend the syntax for a list comprehension to include a condition:

[0, 2, 4, 6, 8]

```
my_list = [x for x in range(0, 10) if x % 2 == 0]
```

- ▶ The general format is as follows:

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

- ▶ Similar to regular list comprehensions, except now we might **not** perform the expression on every member of the list.
- ▶ We first check each member of the list to see if it satisfies a filter condition. Those list members that return False for the filter condition will be omitted from the list before the list comprehension is evaluated.



Examples:

- ▶ Examples:

```
vec = [2, 4, 6]
my_list = [3*x for x in vec if x > 3]
```

[12, 18]

- ▶ Only 4 and 6 satisfy the filter condition.
- ▶ $3*4, 3*6$
- ▶ So, only 12 and 18 are produced.

```
vec = [2, 4, 6]
my_list = [3*x for x in vec if x < 2]
```

[]

- ▶ Get all the factors of a number:

```
n = 6
my_list = [x for x in range(1, n+1) if n%x == 0]
```

[1, 2, 3, 6]



(Filtered List)

- ▶ Example:

```
name = 'Andrew Luxton-Reilly'
vowels = 'aeiou'
my_list = [c for c in name if c not in vowels]
```

['A', 'n', 'd', 'r', 'w', 'l', 'x', 't', 'n', 'l', 'R', 'l', 'l', 'y']



Exercise 2

- ▶ Create a list of all the vowels in the string:
 - ▶ Example: “solidarity”
 - ▶ Output: ['o', 'i', 'a', 'i']

```
word = 'solidarity'  
vowels = 'aeiou'  
...
```



Summary: Features of lists

- ▶ Information in a list is stored contiguously in memory
 - ▶ location of the information can be calculated
 - ▶ location = start of the list + index * size of each element
- ▶ Efficiency issues
 - ▶ It takes the same time to *access* any of the elements
 - ▶ Slow to move elements around (i.e. add and delete elements from within the list)



Exercise

- ▶ Write a list comprehension that generates a list of tuples.
 - ▶ The tuple contains the number (even number between 0 to 9), the square of the number, and the cube of the number.

```
[(0, 0, 0), (2, 4, 8), (4, 16, 64),  
(6, 36, 216), (8, 64, 512)]
```



Exercise 3

- ▶ What is the output of the following code fragment?

```
li = [(x,y) for x in range(3) for y in range(2)]  
print(li)
```