

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding Feb 4 2020, Lecture 5

Agenda

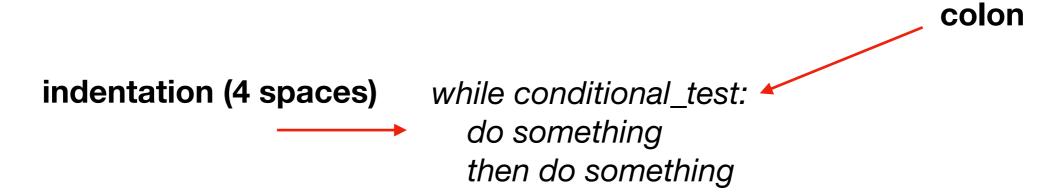
- Agenda:
 - Quick review of concepts from last lecture
 - Functions and modules

Python loops

- if statements allow you to execute different piece of code based on the different situations (conditional test)
- Loops allow you to execute the same piece of code multiple times
- Python has two primitive loop commands
 - while loops
 - for loops

While loop

while loop syntax



- It will keep execute the code block as long as the conditional test is true.
 - usually you will need to modify the the values used in the conditional test once some conditions are met

for loop

- For-each is Python's only form of for loop, this is less like the for keyword in other programming languages.
- A for loop steps through each of the items in a collection type (list, dictionary, etc) or any other type of object which is "iterable" (remember when we call .keys() method of a dictionary)
- Often used with lists and dictionaries

```
indentation (4 spaces)

for <each item> in <collection>:

<statements>
```

while/for loops

- Using break to exit a loop
 - To exit a loop immediately without running any remaining code in the loop
- Using continue in a loop
 - Rather than breaking out of a loop entirely without executing the rest of its code, you can use the continue statement to return to the beginning of the loop based on the result of a conditional test

Avoid infinite loops

Avoid infinite loops when using While

- Might not be an issue using for loop
 - It iterates through each element in a collection (or any object that is iterable) until the end.

for loop Demo



List Comprehensions

- A powerful feature of the Python language
 - Generate a new list by applying a function to every member of an original list
 - Python programmers use list comprehensions extensively. You'll see many of them in real code

[expression for item in list]

List Comprehensions

```
li = [3, 6, 2, 7]
new_list = [elem*2 for elem in li]
print(new_list)

loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
[6, 12, 4, 14]
Process finished with exit code 0
```

[expression for item in list]

- Where expression is some calculation or operation acting upon the variable item.
- For each member of the list, the list comprehension
 - sets item equal to that member, and
 - calculates a new value using expression.
- It then collects these new values into a list which is the return value of the list comprehension.

Filtered List Comprehensions

[expression for item in list if filter]

- Filter determines whether expression is performed on each member of the list
- When processing each element of list, first check if it satisfies the filter condition
- If the filter condition returns False, that elements is omitted from the list before the list comprehension is evaluated.

Filtered List Comprehensions

[expression for item in list if filter]

```
li = [3, 6, 2, 7]
new_list = [elem*2 for elem in li if elem > 3]
print(new_list)

loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
[12, 14]
Process finished with exit code 0
```

Nested List Comprehensions

 Since list comprehensions take a list as input and produce a list as output, they are easily nested:

Self-test: what do you think the nested_li will be?

```
li = [3, 2, 4, 1]
nested_li = [elem*2 for elem in
  [item+1 for item in li]]
```

Nested List Comprehensions

```
li = [3, 2, 4, 1]

nested_li = [elem*2 for elem in

[item*1 for item in li]]

print(nested_li)

loops ×

/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py

[8, 6, 10, 4]

Process finished with exit code 0
```

- The inner comprehension produces: [4, 3, 5, 2]
- So, the outer one produces: [8, 6, 10, 4]