Control flow: loops

Ruth Anderson
UW CSE 160
Spring 2018
Temperature conversion chart

Recall exercise from previous lecture

```python
fahr = 30
cent = (fahr - 32) / 9.0 * 5
print fahr, cent
fahr = 40
cent = (fahr - 32) / 9.0 * 5
print fahr, cent
fahr = 50
cent = (fahr - 32) / 9.0 * 5
print fahr, cent
fahr = 60
cent = (fahr - 32) / 9.0 * 5
print fahr, cent
fahr = 70
cent = (fahr - 32) / 9.0 * 5
print fahr, cent
print "All done"
```

Output:

30 -1.11
40 4.44
50 10.0
60 15.56
70 21.11
All done
A better way to repeat yourself:

```python
for f in [30, 40, 50, 60, 70]:
c = (f - 32) / 9.0 * 5
print f, c
```

Output:

- 30 -1.11
- 40 4.44
- 50 10.0
- 60 15.56
- 70 21.11

"All done"

Note: f and c are not good variable names!
Loop Examples

```python
for num in [2, 4, 6]:
    print num
```

```python
for i in [1, 2, 3]:
    print "Hi there!"
```

```python
for char in "happy":
    print char
```
How a loop is executed: Transformation approach

Idea: convert a `for` loop into something we know how to execute

1. Evaluate the sequence expression
2. Write an assignment to the loop variable, for each sequence element
3. Write a copy of the loop after each assignment
4. Execute the resulting statements

```
for i in [1, 4, 9]:
    print i
```

```
i = 1
print i
i = 4
print i
i = 9
print i
```

State of the computer:
```
i: 4
```

Printed output:
```
1
4
9
```
How a loop is executed: Direct approach

1. Evaluate the sequence expression
2. While there are sequence elements left:
   a) Assign the loop variable to the next remaining sequence element
   b) Execute the loop body

for i in [1, 4, 9]:
    print i

Current location in list

State of the computer:

Printed output:

1
4
9
The body can be multiple statements

Execute whole body, then execute whole body again, etc.

```python
for i in [3, 4, 5]:
    print "Start body"
    print i
    print i * i
```

Convention: often use i or j as loop variable if values are integers

This is an exception to the rule that variable names should be descriptive
The body can be multiple statements

Execute whole body, then execute whole body again, etc.

```python
for i in [3, 4, 5]:
    print "Start body"
    print i
    print i * i
```

### Output:
```
Start body
3
9
Start body
4
16
Start body
5
25
```

### NOT:
```
Start body
Start body
Start body
3
4
5
Start body
9
16
```

Convention: often use i or j as loop variable if values are integers

This is an exception to the rule that variable names should be descriptive
Indentation is significant

- Every statement in the body must have exactly the same indentation
- That’s how Python knows where the body ends

```python
for i in [3, 4, 5]:
    print "Start body"
    print i
    print i*i
```

- Compare the results of these loops:

```python
for f in [30, 40, 50, 60, 70]:
    print f, (f - 32) / 9.0 * 5
print "All done"
```

```python
for f in [30, 40, 50, 60, 70]:
    print f, (f - 32) / 9.0 * 5
print "All done"
```
The range function

A typical for loop does not use an explicit list:
```python
for i in range(5):
    ... body ...
```

- `range(5)` produces `[0, 1, 2, 3, 4]`
- `range(1, 5)` produces `[1, 2, 3, 4]`
- `range(1, 10, 2)` produces `[1, 3, 5, 7, 9]`

**Upper limit** *(exclusive)*
**Lower limit** *(inclusive)*
**step (distance between elements)*

Produce the list:
[0, 1, 2, 3, 4]
Some Loops

# Sum of a list of values, what values?
result = 0
for element in range(5):
    result = result + element
print "The sum is: " + str(result)

# Sum of a list of values, what values?
result = 0
for element in range(5, 1, -1):
    result = result + element
print "The sum is: ", result

# Sum of a list of values, what values?
result = 0
for element in range(0, 8, 2):
    result = result + element
print "The sum is: ", result

# Sum of a list of values, what values?
result = 0
size = 5
for element in range(size):
    result = result + element
print "When size = " + str(size) + " result is " + str(result)
How to process a list:
One element at a time

- A common pattern when processing a list:
  ```python
  result = initial_value
  for element in list:
    result = updated result
  use result
  ```

- `initial_value` is a correct result for an empty list
- As each element is processed, `result` is a correct result for a prefix of the list
- When all elements have been processed, `result` is a correct result for the whole list

```python
# Sum of a list
result = 0
for element in mylist:
    result = result + element
print(result)
```
Examples of list processing

- **Product of a list:**
  ```python
  result = 1
  for element in mylist:
    result = result * element
  ```

- **Maximum of a list:**
  ```python
  curr_max = mylist[0]
  for element in mylist:
    curr_max = max(curr_max, element)
  ```

- **Approximate the value 3 by**
  ```latex
  \[1 + \frac{2}{3} + \frac{4}{9} + \frac{8}{27} + \frac{16}{81} + \ldots = \left(\frac{2}{3}\right)^0 + \left(\frac{2}{3}\right)^1 + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3 + \ldots + \left(\frac{2}{3}\right)^{10}\]
  ```
  ```python
  result = 0
  for element in range(11):
    result = result + (2.0/3.0)**element
  ```
Nested Loops

```python
for i in [1, 2, 3]:
    print "Before j loop i is", i
    for j in [50, 100]:
        print "j is", j
```

What is the output?
More Nested Loops

How many statements does this loop contain?

```python
for i in [0, 1]:
    print "Outer", i
    for j in [2, 3]:
        print " Inner", j
        print " Sum", i + j
    print "Outer", i
```

What is the output?

See in python tutor
More Nested Loops

How many statements does this loop contain?

```python
for i in [0, 1):
    print "Outer", i
    for j in [2, 3]:
        print " Inner", j
        print " Sum", i + j
    print "Outer", i
```

What is the output?

Output:
- Outer 0
- Inner 2
- Sum 2
- Inner 3
- Sum 3
- Outer 0
- Outer 1
- Inner 2
- Sum 3
- Inner 3
- Sum 4
- Outer 1

"nested" loop body: 2 statements
- loop body: 3 statements

See in python tutor
Understand loops through the transformation approach

Key idea:
1. Assign each sequence element to the loop variable
2. Duplicate the body

```python
for i in [0, 1]:
    i = 0
    print "Outer", i
def j in [2, 3]:
    for j in [2, 3]:
        print "Inner", j
    i = 1
    print "Outer", i
    for j in [2, 3]:
        print "Inner", j
i = 0
print "Outer", i
for j in [2, 3]:
    j = 2
    print "Inner", j
    j = 3
    print "Inner", j
i = 1
print "Outer", i
for j in [2, 3]:
    print "Inner", j
```
Test your understanding of loops

Puzzle 1:

```python
for i in [0, 1]:
    print i
print i
```

Output:

```
1
```

Puzzle 2:

```python
i = 5
for i in []:
    print i
```

Puzzle 3:

```python
for i in [0, 1]:
    print "Outer", i
for i in [2, 3]:
    print "Inner", i
print "Outer", i
```

Reusing loop variable (don’t do this!)
Test your understanding of loops

Puzzle 1:
```python
for i in [0, 1]:
    print i
print i
```

Output:
```
0
1
```

Puzzle 2:
```python
i = 5
for i in []:  
    print i
```

(no output)

Puzzle 3:
```python
for i in [0, 1]:
    print "Outer", i
for i in [2, 3]:
    print "Inner", i
print "Outer", i
```

Output:
```
Outer 0
Inner 2
Outer 1
Inner 2
Outer 3
Inner 3
Outer 3
```
Fix this loop

# Goal: print 1, 2, 3, ..., 48, 49, 50
for tens_digit in [0, 1, 2, 3, 4]:
    for ones_digit in [1, 2, 3, 4, 5, 6, 7, 8, 9]:
        print tens_digit * 10 + ones_digit

What does it actually print?
How can we change it to correct its output?

Moral: Watch out for edge conditions (beginning or end of loop)
Some Fixes

```python
for tens_digit in [0, 1, 2, 3, 4]:
    for ones_digit in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
        print tens_digit * 10 + ones_digit + 1

for tens_digit in [0, 1, 2, 3, 4]:
    for ones_digit in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
        print tens_digit * 10 + ones_digit

for ones_digit in [1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print ones_digit
for tens_digit in [1, 2, 3, 4]:
    for ones_digit in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
        print tens_digit * 10 + ones_digit
print 50
```
for size in [1, 2, 3, 4]:
    print "size is " + str(size)
    for element in range(size):
        print "element is " + str(element)
for size in [1, 2, 3, 4]:
    result = 0
    for element in range(size):
        result = result + element
    print "size=" + str(size) + " result=" + str(result)
print " We are done!"

What happens if we move \texttt{result = 0} to be the first line of the program instead?