Control flow

Ruth Anderson
UW CSE 160
Winter 2017
Repeating yourself

Making decisions
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:

<table>
<thead>
<tr>
<th>Fahrenheit</th>
<th>Celsius</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>-1.11</td>
</tr>
<tr>
<td>40</td>
<td>4.44</td>
</tr>
<tr>
<td>50</td>
<td>10.0</td>
</tr>
<tr>
<td>60</td>
<td>15.56</td>
</tr>
<tr>
<td>70</td>
<td>21.11</td>
</tr>
</tbody>
</table>

All done
Temperature conversion chart

A better way to repeat yourself:

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

Output:

```
30 -1.11
40 4.44
50 10.0
60 15.56
70 21.11
All done
```
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

```python
for i in [1, 4, 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

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

Current location in list

State of the computer:

Printed output:

```
i: 4
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
9
16
25
```

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
```

Error!

```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"
```

See in python tutor
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?
Nested Loops

How many statements does this loop contain?

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

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

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

What is the output?
Understand loops through the transformation approach

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

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

for i in [0,1]:
   i = 0
   print "Outer", i
   for j in [2,3]:
      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
```

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
```

Output:
```
1
Reusing loop variable
(don’t do this!)
```
Test your understanding of loops

Puzzle 1:

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

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
```

Output:

```
0
1
(no output)
```

Reusing loop variable (don’t do this!)
# Goal: print 1, 2, 3, ..., 48, 49, 50

```python
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)
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
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 \([1, 3, 5, 7, 9]\)
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)
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 result = 0 to be the first line of the program instead?
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

# Sum of a list
```python
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 \( 1 + \frac{2}{3} + \frac{4}{9} + \frac{8}{27} + \frac{16}{81} + \ldots \) = \((\frac{2}{3})^0 + (\frac{2}{3})^1 + (\frac{2}{3})^2 + (\frac{2}{3})^3 + \ldots + (\frac{2}{3})^{10}\)
  ```python
  result = 0
  for element in range(11):
    result = result + (2.0/3.0)**element
  ```

See in python tutor
```python
result = initial_value
for element in list:
  result = updated result
```
Making decisions

• How do we compute absolute value?
  abs(5) = 5
  abs(0) = 0
  abs(-22) = 22
Absolute value solution

If the value is negative, negate it. Otherwise, use the original value.

val = -10

# calculate absolute value of val
if val < 0:
    result = -val
else:
    result = val

print result

Another approach that does the same thing without using result:

val = -10

if val < 0:
    print -val
else:
    print val

In this example, result will always be assigned a value.
Absolute value solution

As with loops, a sequence of statements could be used in place of a single statement:

```python
val = -10

# calculate absolute value of val
if val < 0:
    result = -val
    print "val is negative!"
    print "I had to do extra work!"
else:
    result = val
    print "val is positive"
print result
```

See in python tutor
val = 0

# calculate absolute value of val
if val < 0:
    print "val is negative"
    print val
    result = - val
elif val == 0:
    print "val is zero"
    print val
    result = val
else:
    print "val is positive"
    print val
    result = val

print result
Another absolute value solution

What happens here?

```python
val = 5

# calculate absolute value of val
if val < 0:
    result = -val
    print "val is negative!"
else:
    for i in range(val):
        print "val is positive!"
    result = val
print result
```

See in python tutor
Another if

It is **not required** that anything happens...

```python
val = -10

if val < 0:
    print "negative value!"
```

What happens when `val = 5`?
The if body can be any statements

Execution gets here only if “height > 100” is false

Execution gets here only if “height > 100” is false AND “height > 50” is true
# height is in km
if height > 100:
    print "space"
else:
    if height > 50:
        print "mesosphere"
    else:
        if height > 20:
            print "stratosphere"
        else:
            print "troposphere"
# height is in km
if height > 100:
    print "space"
else:
    if height > 50:
        print "mesosphere"
    else:
        if height > 20:
            print "stratosphere"
        else:
            print "troposphere"
Version 2

if height > 50:
    if height > 100:
        print "space"
    else:
        print "mesosphere"
else:
    if height > 20:
        print "stratosphere"
    else:
        print "troposphere"
Version 3 (Best)

```python
if height > 100:
    print "space"
elif height > 50:
    print "mesosphere"
elif height > 20:
    print "stratosphere"
else:
    print "troposphere"
```

ONE of the print statements is guaranteed to execute: whichever condition it encounters first that is true.
Order Matters

# version 3
if height > 100:
    print "space"
elif height > 50:
    print "mesosphere"
elif height > 20:
    print "stratosphere"
else:
    print "troposphere"

# broken version 3
if height > 20:
    print "stratosphere"
elif height > 50:
    print "mesosphere"
elif height > 100:
    print "space"
else:
    print "troposphere"

Try height = 72 on both versions, what happens?
In this case it is possible that nothing is printed at all, when?
What Happens Here?

# height is in km
if height > 100:
    print "space"
if height > 50:
    print "mesosphere"
if height > 20:
    print "stratosphere"
else:
    print "troposphere"

Try height = 72
The then clause or the else clause is executed

```python
speed = 54
limit = 55
if speed <= limit:
    print "Good job!"
else:
    print "You owe $", speed/fine
```

What if we change speed to 64?