Control Flow

BBM 101 - Introduction to Programming I

Hacettepe University
Fall 2016

Fuat Akal, Aykut Erdem, Erkut Erdem

Slides based on material prepared by Ruth Anderson, Michael Ernst and Bill Howe in the course CSE 140
University of Washington
Repeating yourself

Making decisions
Recall the exercise from the 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 15.55
70 21.11
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:

<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>15.55</td>
</tr>
<tr>
<td>70</td>
<td>21.11</td>
</tr>
</tbody>
</table>

All done
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)
```

State of the computer:

Printed output:
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:
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
```

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 in Loop 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)
print("All done")
```

- 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 Body can be Multiple Statements

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
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)
    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)
```
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

# Goal: print 1, 2, 3, ..., 48, 49, 50

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 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

• Analyze each of the above
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 []:  # Empty list
    print(i)
```
Output: (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
        Inner 3
        Outer 3
        Outer 1
        Inner 2
        Inner 3
        Outer 3

Reusing loop variable (don’t do this!)
The Range Function

As an implicit list:

```python
for i in range(5):
    ... body ...
```

The list $[0,1,2,3,4]$:

- Upper limit (exclusive) $= 5$
- Lower limit (inclusive) $= 0$
- Step (distance between elements) $= 1$

- $\text{range}(5) = [0,1,2,3,4]$
- $\text{range}(1,5) = [1,2,3,4]$
- $\text{range}(1,10,2) = [1,3,5,7,9]$
Decomposing a List Computation

• To compute a value for a list:
  – Compute a partial result for all but the last element
  – Combine the partial result with the last element

Example: sum of a list:

\[ [ 3, 1, 4, 1, 5, 9, 2, 6, 5 ] \]

\[
\begin{align*}
\text{sum(List a)} &= \text{sum(List b)} + 5 \\
\text{sum(List b)} &= \text{sum(List c)} + 6 \\
\vdots \\
\text{sum(List y)} &= \text{sum(List z)} + 3 \\
\text{sum(empty list)} &= 0
\end{align*}
\]
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
  ```
  # Sum of a list
  ```python
  result = 0
  for element in mylist:
      result = result + element
  print(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
Some Loops

# Sum of a list of values, what values?
result = 0
for element in range(5):  # [0,1,2,3,4]
    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) + " , the result is " + str(result))

Traceback (most recent call last):
  File "C:\Users\Vahid\PycharmProjects\untitled\main.py", line 7
    print("The sum is: " + str(result))
TypeError: Can't convert 'int' object to str implicitly
Process finished with exit code 1
Some More Loops

```python
for size in [1, 2, 3, 4]:
    result = 0
    print("size=" + str(size))
    for element in range(size):
        result = result + element
        print(" adding " + str(element)+", result so far=" + str(result))
    print("Done. size=" + str(size) + " result=" + str(result))
print("All done!")
```

Output?

```
for size in [1, 2, 3, 4]:
    result = 0
    print("size=1")
    for element in range(size):
        result = result + element
        print(" adding " + str(element)+", result so far=" + str(result))
    print("Done. size=1 result=0")
size=2
    adding 0, result so far=0
    adding 1, result so far=1
Done. size=2 result=1
size=3
    adding 0, result so far=0
    adding 1, result so far=1
    adding 2, result so far=3
Done. size=3 result=3
size=4
    adding 0, result so far=0
    adding 1, result so far=1
    adding 2, result so far=3
    adding 3, result so far=6
Done. size=4 result=6
All done!
```
Some More Loops

```python
result = 0
for size in [1, 2, 3, 4]:
    result = 0
    print("size=" + str(size))
    for element in range(size):
        result = result + element
        print(" adding " + str(element)+", result so far=" + str(result))
    print("Done. size=" + str(size) + " result=" + str(result))
print("All done!")
```

What happens if we move `result = 0` to be the first line of the program instead?
Examples of List Processing

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

• Maximum of a list:
  ```python
  result = mylist[0]
  for element in mylist:
    result = max(result, 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
  ```

The first element of the list (counting from zero) can be calculated as:

```python
result = initial_value
for element in list:
    result = updated result
```
Exercise with Loops

• Write a simple program to add values between two given inputs a, b
• e.g., if a=5, b=9, it returns sum of (5+6+7+8+9)
• **Hint:** we did some ‘algorithmic thinking’ and ‘problem solving’ here!

```python
a=5
b=9
total = 0
for x in range(a, b+1):
    total += x
print(total)
```
Another Type of Loops

- The **while** loop is used for repeated execution as long as an expression is true

```python
n = 100
s = 0
counter = 1
while counter <= n:
    s = s + counter
    counter += 1

print("Sum of 1 until %d: %d" % (n,s))
```

Sum of 1 until 100: 5050
How do we compute absolute value?

\[
\begin{align*}
\text{abs}(5) &= 5 \\
\text{abs}(0) &= 0 \\
\text{abs}(-22) &= 22
\end{align*}
\]
Absolute Value Solution

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

```python
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`:

```python
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 inside an if 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)
```
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)
```
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

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

Written differently, but more efficient!

```python
# height is in km
if height > 100:
    print("space")
elif height > 50:
    print("mesosphere")
elif height > 20:
    print("stratosphere")
else:
    print("troposphere")
```

Execution gets here only if "height > 100" is false

Execution gets here only if "height > 100" is false AND "height > 50" is true

The diagram shows the different layers of the Earth's atmosphere:
- Troposphere
- Stratosphere
- Mesosphere
- Space

The scale indicates that "28 km" is above the Earth.
# height is in km

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

Execution gets here only if "height <= 100" is true

Execution gets here only if "height <= 100" is true AND "height > 50" is true

<table>
<thead>
<tr>
<th>troposphere</th>
<th>stratosphere</th>
<th>mesosphere</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>80</td>
<td>90</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

km above earth
# height is in km
if height > 100:
    print("space")
else:
    if height > 50:
        print("mesosphere")
    else:
        if height > 20:
            print("stratosphere")
        else:
            print("troposphere")
if height > 50:
    if height > 100:
        print("space")
    else:
        print("mesosphere")
else:
    if height > 20:
        print("stratosphere")
    else:
        print("troposphere")
Version 3

```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?
Version 3

# incomplete version 3
if height > 100:
    print("space")
elif height > 50:
    print("mesosphere")
elif height > 20:
    print("stratosphere")

In this case it is possible that nothing is printed at all, when?
# What Happens Here?

```python
# 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 \textit{or} the else Clause is Executed

\begin{verbatim}
speed = 54
limit = 55
if speed <= limit:
    print("Good job, safe driver!")
else:
    print("You owe $", speed/fine)
\end{verbatim}

What if we change speed to 64?
The break Statement

• The **break** statement terminates the current loop and resumes execution at the next statement

```python
for letter in 'hollywood':
    if letter == 'l':
        break
    print('Current Letter :', letter)
```

Current Letter : h
Current Letter : o
The continue Statement

• The `continue` statement in Python returns the control to the beginning of the while loop.

```python
for letter in 'hollywood':
    if letter == 'l':
        continue
    print ('Current Letter :', letter)
```

Current Letter : h
Current Letter : o
Current Letter : y
Current Letter : w
Current Letter : o
Current Letter : o
Current Letter : d