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

Temperature Conversion Chart

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
```

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 15.55
70 21.11
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

State of the computer: Printed output:

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

```
i: 1
i: 4
i: 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

State of the computer: Printed output:

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

```
i: 1
i: 4
i: 9
```

The Body can be Multiple Statements

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

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

Output: NOT:

```
 Output:   NOT:
---      ---
Start body  Start body
Start body  Start body
Start body  Start body
Start body  Start body
3        Start body
4        Start body
9        Start body
```

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

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

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

- Compare the results of these loops:
- This is an exception to the rule that variable names should be descriptive
The Body can be Multiple Statements

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)
```

What is the output?

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

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)

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

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)
```

- 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
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)
```

The Range Function

As an implicit list:
```python
for i in range(5):
    ... body ...
```
```
range(5) = [0,1,2,3,4]
range(1,5) = [1,2,3,4]
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 ]
```
```
List z
List y
List c
List b
List a
```

• A common pattern when processing a list:
  ```python
  result = initial_value
  for element in list:
      result = updated result
      use result
  # Sum of a list
  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

```python
# 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?
size = 5
for element in range(size):
    result = result + element
print("When size = " + str(size) + ", the result is "+ str(result))
```

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("Done. size= " + str(size) + ", result=" + str(result))
print("All done!")
```

Examples of List Processing

- Product of a list:
  ```python
  result = 0
  for element in mylist:
      result = result * element
  print("Done. size= " + str(size) + ", result=" + str(result))
  print("All done!")
  ```

- Maximum of a list:
  ```python
  result = mylist[0]
  for element in mylist:
      result = max(result, element)
  print("Done. size= " + str(size) + ", result=" + str(result))
  print("All done!")
  ```

- Approximate the value 3 by $1 + 2/3 + 4/9 + 8/27 + 16/81 + \ldots = (2/3)^0 + (2/3)^1 + (2/3)^2 + (2/3)^3 + \ldots + (2/3)^10$
  ```python
  result = 0
  for element in range(11):
      result = result + (2.0/3.0)**element
  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?
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))
```

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.

```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)
```

Another if

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

```python
val = -10
if val < 0:
    print("negative value!")
```

What happens when val = 5?

```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)
```

The if Body can be Any Statements

```python
# height is in km
if height > 100:
    print("space")
else:
    if height > 50:
        print("mesosphere")
    elif 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")
```
Version 1

```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")
```

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

Version 2

```python
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")
```
Order Matters

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

### Broken Version 3

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

Try height = 72 on both versions, what happens?

### 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

### 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?

### The then Clause or the else Clause is Executed

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

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
```