Control flow: Loops

Andrew S Fitz Gibbon
UW CSE 160
Winter 2022
Exercise: Convert temperatures

- Make a temperature conversion chart, from Fahrenheit to Centigrade, for these Fahrenheit values: 30, 40, 50, 60, 70

- Output (approximate):
  
<table>
<thead>
<tr>
<th>Fahrenheit</th>
<th>Centigrade</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

- Hint: \[ \text{cent} = \frac{\text{fahr} - 32}{9.0} \times 5 \]
One possible Python program that solves this:

```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
```
Copy and Paste Problems

• Error prone
• Can take a long time (luckily this list only had 5 values in it!)
• What about ...
  – Modifications: I decide I want to change the output format?
  – Bugs: I made a mistake in the formula?
  – Readability: Is it obvious to a human reader that all 5 chunks of code are identical without looking carefully?
For each fahr, do “this”

• Where “this” is:

  cent = (fahr - 32) / 9.0 * 5
  print(fahr, cent)

• Would be nice if we could write “this” just once
  – Easier to modify
  – Easier to fix bugs
  – Easier for a human to read
A for loop

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

- Would be nice if we could write “this” just once
  - Easier to modify
  - Easier to fix bugs
  - Easier for a human to read
A better way to repeat yourself:

```python
for fahr in [30, 40, 50, 60, 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
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

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

State of the computer:

Printed output:

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

Current location in list

```python
for i in [1,4,9]:
  print(i)
```
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
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
print(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): cycles through [0, 1, 2, 3, 4]

range(1, 5): cycles through [1, 2, 3, 4]

range(1, 10, 2): cycles through [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))
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
Examples of list processing

- Product of a list:
  ```python
def product(mylist):
    result = 1
    for element in mylist:
        result *= element
    return result
```

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

- Approximate the value 3 by
  $$1 + \frac{2}{3} + \frac{4}{9} + \frac{8}{27} + \frac{16}{81} + \ldots = \left(\frac{2}{3}\right)^0 + \left(\frac{2}{3}\right)^1 + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3 + \ldots + \left(\frac{2}{3}\right)^{10}$$
  ```python
def approximate_3(initial_value=0):
    result = 0
    for element in range(11):
        result += (2.0/3.0)**element
    return result
```

The first element of the list (counting from zero)

See in python tutor
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?
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
- Outer 1
  - Inner 2
  - Sum 3
- Outer 0
- Outer 1
  - Inner 3
  - 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]:
    print("Outer", i)
    for j in [2, 3]:
        print(" Inner", j)
        i = 1
    print("Outer", i)
    for j in [2, 3]:
        print(" Inner", j)
        j = 2
        j = 3
        print(" Inner", j)
    i = 0
    print("Outer", i)
```

See in python tutor
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)
```

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

- Puzzle 1:
  
- Puzzle 2:
  
- Puzzle 3:
  
  - Outer 0
  - Inner 2
  - Inner 3
  - Outer 3
  - Outer 1
  - Inner 2
  - Inner 3
  - Outer 3

See in python tutor
Some More Loops

```python
for size in [1, 2, 3, 4]:
    print("size is " + str(size))
    for element in range(size):
        print("element is " + str(element))
```
Even More Loops

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!")
print("result is", result)

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

See in python tutor
Loops over Strings

for letter in "hello":
    print(letter)

my_string = "CSE 160"
for letter in my_string:
    print(letter)

count = 0
for letter in my_string:
    count = count + 1
print(count)