

# Control flow: Loops

Ruth Anderson

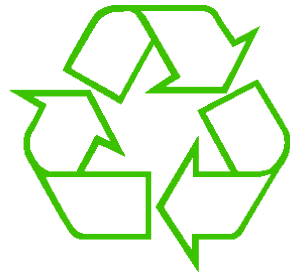
UW CSE 160

Autumn 2020

# Exercise: Convert temperatures

- Make a temperature conversion chart, from Fahrenheit to Centigrade, for these Fahrenheit values: 30, 40, 50, 60, 70
- Output (approximate):  
30 -1.11  
40 4.44  
50 10.0  
60 15.56  
70 21.11  
All done
- Hint:  $\text{cent} = (\text{fahr} - 32) / 9.0 * 5$

# Temperature conversion chart



One possible Python program that solves this:

[See in python tutor](#)

```
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)
```

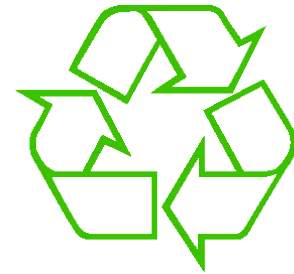
- Would be nice if we could write “`this`” **just once**
  - Easier to **modify**
  - Easier to **fix bugs**
  - Easier for a human to **read**

# A for loop

```
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**

# for Loop Explained



A better way to repeat yourself:

[See in python tutor](#)

for loop

loop variable or iteration variable

A list  
(sequence expression can be any sequence type e.g. string)

Colon is required

Loop *body* is indented

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

```
print("All done")
```

Indentation is significant!

Excutes the *body* 5 times:

- once with `fahr = 30`
- once with `fahr = 40`
- ...

Output:

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

# Loop Examples

[See in python tutor](#)

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

Prints the values  
of sequence

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

Does not use values  
of sequence

```
for char in "happy":  
    print(char)
```

sequence is a string

Prints the values  
of sequence



# 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

[See in python tutor](#)

```
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:

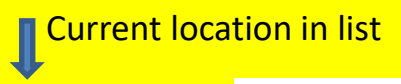
```
1  
4  
9
```

# 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

A yellow box labeled "Current location in list" has a blue arrow pointing down to the element '4' in the list [1, 4, 9] of the code snippet above.

State of the  
computer:

```
i: 4
```

Printed output:

```
1  
4  
9
```

# The body can be multiple statements

[See in python tutor](#)

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

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

} loop body:  
3 statements

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.

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

} loop body:  
3 statements

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

[See in python tutor](#)

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

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

Error! `print(i)`  
`print(i*i)`

- Compare the results of these loops:

```
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")
```

# The range function

A typical for loop does not use an explicit list:

```
for i in range (5) :
```

```
... body ...
```

Upper limit  
(*exclusive*)

Produces a range  
object

**range (5)** → will loop through [0, 1, 2, 3, 4]

Lower limit  
(*inclusive*)

**range (1, 5)** → will loop through [1, 2, 3, 4]

step (distance  
between elements)

**range (1, 10, 2)** → will loop through [1, 3, 5, 7, 9]

# Some Loops

[See in python tutor](#)

```
# 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:

```
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



# Examples of list processing

- Product of a list:

```
result = 1
for element in mylist:
    result = result * element
```

```
result = initial_value
for element in list:
    result = updated result
```

- Maximum of a list:

```
curr_max = mylist[0]
for element in mylist:
    curr_max = max(curr_max, element)
```

The first element of the list (counting from zero)

- Approximate the value 3 by  $1 + 2/3 + 4/9 + 8/27 + 16/81 + \dots$   
 $= (2/3)^0 + (2/3)^1 + (2/3)^2 + (2/3)^3 + \dots + (2/3)^{10}$

```
result = 0
for element in range(11):
    result = result + (2.0/3.0)**element
```

# Nested Loops

```
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

[See in python tutor](#)

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?

# More Nested Loops

[See in python tutor](#)

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)
```

"nested"  
loop body:  
2 statements

loop body:  
3 statements

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

What is the output?

# Understand loops through the transformation approach

[See in python tutor](#)

Key idea:

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

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

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)
j = 2
print(" Inner", j)
j = 3
print(" Inner", j)
i = 1
print("Outer", i)
for j in [2, 3]:
    print(" Inner",21j)
```

# Test your understanding of loops

Puzzle 1:

```
for i in [0, 1]:  
    print(i)  
print(i)
```

Output:

Puzzle 2:

```
i = 5  
for i in []:  
    print(i)
```

Puzzle 3:

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

inner loop body

outer loop body

# Test your understanding of loops

Puzzle 1:

```
for i in [0, 1]:  
    print(i)  
print(i)
```

Output:

0  
1  
1

Puzzle 2:

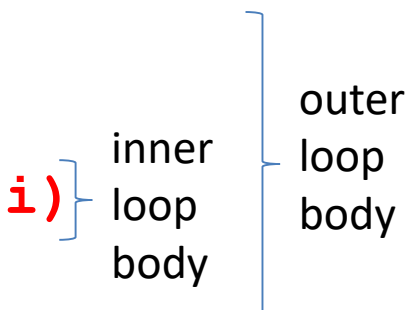
```
i = 5  
for i in []:  
    print(i)
```

(no output)

Puzzle 3:

```
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!)



Outer 0  
Inner 2  
Inner 3  
Outer 3  
Outer 1  
Inner 2  
Inner 3  
Outer 3

# Some More Loops

```
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!")
```

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

# Fix this loop

[See in python tutor](#)

```
# 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

[See in python tutor](#)

```
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)
```

# Loops over Strings

[See in python tutor](#)

```
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)
```