for Loops

CSC120

Mark Kazakevich

Repeating code

- We've talked a bit about how functions are used to minimize repetitive code
- If we have a block of code that we want to be able to bring up and use any time, and with different data, we can define a function and put that block of code in it
- Today we're going to look at another way we can deal with code that needs to be repeated

Repeating code

- Often there will be times where writing a function isn't going to be enough to get rid of your need to repeat code.

- For example:
  - What if you wanted to call a function **1000 times**?
  - You can write a function that calls that function 1000 times, but that is going to require a file that is **1000 lines long**
  - Not very efficient...

Repeating code

- More relevant example for us:
  - What if we wanted to do something with each **individual** character of a string?
  - If we had a string `s = 'hello'`
    right now we would have to index every element manually:
    s[0], s[1], s[2], … and so on.
  - Not very efficient…

- We want a way to get to all of the values individually without writing out the index numbers each time

Loops

- In order to help us with this problem, we're going to introduce a new concept: **Loops**
- Simply: Loops allow us to repeat a block of code
- Like if-statements, this is another way to change the program flow of Python.
- Today we will be looking at one type of loop in Python, the **for** loop

for loops

- A **for** loop is a statement that allows us to repeat code a set number of times.

- The number of times it repeats depends on some ordered set of values
  - For example, the characters in a string from left to right

- for loops take each each value in the ordered set, save it to a variable, and execute the code in the loop
  - It then repeats this process for every element.
- Let's take a closer look

for loop Format

```
for element in set:
    # loop body
```

This block is considered one for loop

Let's talk about what these words all mean

for loop Format

```
for element in set:
    # loop body
```

for

Indicates that this is a for loop statement

for loop Format

```
for element in set:
    # loop body
```

element in set

element is the variable name we are going to give to every element in set as we repeat the code

We call this iterating over a set:
"For every element in set"

for loop Format

```
for element in set:
    # loop body
```

loop body

- These lines of code (which are indented in the for loop), will repeat for every element in the set.
- We can use the value of the variable `element` for the current *iteration* and work with it all the way to the end of the for loop

Let’s see an example

```
s = ‘hello’
for char in s:
   print(char)
```

Let's see an example

```
s = 'hello'
for char in s:
    print(char)
```

for

Indicates that this is a for loop statement

Let's see an example

```
s = 'hello'
for char in s:
    print(char)
```

char in s

We are going to "iterate" over the string `s`.
Every time we repeat the loop body, we will change the value of the variable `char` to be the next character in the string `s`

Let's see an example

```
s = 'hello'
for char in s:
    print(char)
```

`print(char)`

This is the loop body. We are using the variable `char` which is the current character of `s` that we have iterated to.

Running the example

```
s = ‘hello’
for char in s:
   print(char)
```

1st iteration of loop:
Current value of char: **h**

Python shell output after running loop body:

```
h
```

Running the example

```
s = ‘hello’
for char in s:
   print(char)
```

2nd iteration of loop:
Current value of char: **e**

Python shell output after running loop body:

```
h
e
```

Running the example

```
s = ‘hello’
for char in s:
   print(char)
```

3rd iteration of loop:
Current value of char: **l**

and so on..

Python shell output after running loop body:

```
h
e
l
```

End of the string

```
s = ‘hello’
for char in s:
   print(char)
.
# program continues
.
.
.
```

No more characters in s. We’re done! We now move on to the statements after the for loop

```
h
e
l
l
o
```

Something to be careful about

```
k = 5
m = 1

s = ‘hello’
for k in s:
   print(k)

p = k + m
```

- Do not use variables that we assigned outside of the for loop as the name for each element inside the for loop.
- You might need it later, but it will still be assigned to the last element of the set (in this case, ‘o’)

Something to be careful about

```
k = 5
m = 1

s = ‘hello’
for k in s:
   print(k)

p = k + m
```

The last value of k in the loop was ‘o’

The value of p is:

```
p = ‘o’ + 1  #error
```

Rule: use a different variable name in the loop

Looping over a sequence

- We can loop over a sequence of numbers

```
for i in range(4):
    print(i)
```

Shell output:

```
0
1
2
3
```

Convention: Use the variable `i` when looping over a sequence of numbers (with no other context)

Nested loops

- We can put a loop inside a loop

```
for i in range(5):
   for j in range(3)
      print(i)
```

Here we loop through the sequence 0 to 4, and for every element of that sequence, we also loop through 0 to 2.

We will see how this works in Wing

Convention: Use the variables `i` and `j` when looping over a sequence of numbers with nested loops

Examples in Wing