Loops (While and For)

CSE 1310 – Introduction to Computers and Programming

Motivation

- Suppose we want to write a program that does this:
  - Ask the user to input an integer N.
  - Prints out all integers between 0 and N.
- The elements of Python that we have covered so far are not sufficient for writing this program.
- What is missing: the ability to repeat some instructions as many times as we want.

`while` loops

- A **`while`** loop is defined as follows:

```
while boolean_expression:
    line 1
    line 2
    …
    line n
```

- Line 1, line 2, …, line n are called the **body** of the **`while`** loop.

`while` loop execution

```
while boolean_expression:
    line 1
    line 2
    …
    line n
first line after loop
```

- This is how a while loop gets executed:
  - Step 1: evaluate **`boolean_expression`**.
  - Step 2: If the expression is false, go to the first line after the loop.
  - Step 3: If expression is true, execute the body of the while loop, and go back to step 1.

An example of a **`while`** loop

```
number_text = input("enter an integer: ")
number = int(number_text)

i = 0
while (i <= number):
    print(i)
    i = i+1

print("done with the while loop")
```

`while` loops: indentation matters

```
number_text = input("enter an integer: ")
number = int(number_text)

i = 0
while (i <= number):
    print(i)
    i = i+1

    print("done with the while loop")
```

What does this program do?

Designing a **`while`** loop

- When you design a **`while`** loop, you need to make sure that the loop will terminate exactly when needed, not before, and not after.
- You will need to define a test (boolean expression), that determines when to stay in the loop and when to exit.
- You need to update variables within the body of the loop, as needed.

`for` loops (simplest version)

- A **`for`** loop can be defined as follows (note: this definition will be extended when we talk about lists).

```
for variable in range(from, to):
    line 1
    line 2
    …
    line n
```

- Line 1, line 2, …, line n are called the **body** of the **`for`** loop.

`for` loop execution (simplest version)

```
for variable in range(from, to):
    line 1
    line 2
    …
    line n

first line after loop
```

- This is how a for loop gets executed:
  - Step 1: variable = from
  - Step 2: If variable >= to, go to first line after the loop.
  - Step 3: execute the body of the loop.
  - Step 4: update variable to variable + step, and go to step 2

An example of a `for` loop

```
number_text = input("enter an integer: ")
number = int(number_text)

for i in range(0, number+1):
    print(i)

print("done with the for loop")
```

WARNING about using `range`

- If you want to process the integers between X and Y, you need to use range(X, Y+1).
- If you use range(X, Y), the for loop will go up to Y-1, not up to Y.

- This is an extremely common source of bugs.

`for` loops, version 2

- A **`for`** loop can also be defined as follows (note: this definition will be extended when we talk about lists).

```
for variable in range(from, to, step):
    line 1
    line 2
    …
    line n
```

- Line 1, line 2, …, line n are called the **body** of the **`for`** loop.

`for` loop execution

```
for variable in range(from, to, step):
    line 1
    line 2
    …
    line n

first line after loop
```

- This is how a for loop gets executed:
  - Step 1: variable = from
  - Step 2: If **step is positive** and variable >= to, or **step is negative** and variable <= to, go to first line after the loop.
  - Step 3: variable = variable + step
  - Step 4: go to step 2

A `for` loop with a step

```
number_text = input("enter an integer: ")
number = int(number_text)

for i in range(0, number+1, 13):
    print(i)

print()
print("printed all numbers between 0 and", number)
print("that are divisible by 13")
```

A **`for`** loop with a negative step

```
number_text = input("enter an integer: ")
number = int(number_text)

for i in range(number, -1, -1):
    print(i)

print()
print("printed all numbers between", number)
print("and 0 in reverse order")
```

A `for` loop with a negative step

```
number_text = input("enter an integer: ")
number = int(number_text)

for i in range(number, -1, -1):
    print(i)

print()
print("printed all numbers between", number)
print("and 0 in reverse order")
```

Note that the second argument of the range is -1, not 0.

The **`break`** statement

- The **`break`** statement forces termination of the current while loop or for loop.
- Example: print the first number >= N that is divisible by 13.

```
N = int(input("enter an integer: "))
i = N

while True:
  if (i % 13 == 0):
    print("first number >=", N, "divisible by 13 is ", i)
    break
  i = i+1
```

The **`continue`** statement

- The **`continue`** statement skips the rest of the body of the loop and goes directly to the next iteration (or to termination).
- Example: print numbers between 1 and N that are divisible by 13.

```
N = int(input("enter an integer: "))

for i in range(0, N+1):
    if (i % 13 != 0):
        continue
    print(i)
```

`for` loops, general version

- A **`for`** loop, in general, is defined as follows.

```
for variable in set_of_values:
    line 1
    line 2
    …
    line n
```

- Line 1, line 2, …, line n are called the **body** of the **`for`** loop.
- **`set_of_values`** can be, among other things, a string or a **list**. We will cover lists later in the course.

Example 1: `for` loop with a string

```
text = input("enter some text: ")
counter = 0

for i in text:
    print(i)
    if (i == 'a'):
        print("found an 'a'")
        counter = counter+1

print("\nThe letter 'a' appears", counter, "times")
```

Example 1: `for` loop with a string

```
text = input("enter some text: ")
counter = 0

for i in text:
    print(i)
    if (i == 'a'):
        print("found an 'a'")
        counter = counter+1

print("\nThe letter 'a' appears", counter, "times")
```

New elements: string equality, single quote within double quotes, "\n"

Example 2: `for` loop with a string

```
# count the number of vowels in text entered by the user.

text = input("enter some text: ")
vowel_counter = 0

for i in text:
    if (i in 'aeiouyAEIOUY'):
        vowel_counter = vowel_counter + 1

print("\nThe text contains", vowel_counter, "vowels.")
```