Multiple assignment

- It is possible, even advantageous, to make more than one assignment to a variable.

- This makes an existing variable refer to a new value.

- Caution! Do not confuse the assignment '!=' operator with the equality '==' operator.
Updating variables

- Sometimes the change to the value of a variable is dependent on its original value:

  \[ x = x + 1 \]

- The above statement means increment the value of 'x' by one
Looping statements

- Computers are used to perform repetitive tasks

- The 'while' loop is one type of loop

- The 'for' loop is another
The 'while' statement

• Here is an example of a 'while' loop:

```python
i = 10
while i >= 0:
    print(i)
    i = i - 1
print('Blastoff!')
```
The 'while' statement

- The statement can be read as:
  - While 'i' is greater than or equal to zero print 'i' and then decrement 'i'
  - When i is less than zero, print 'Blastoff!'
- More formally:
  1. Evaluate the condition
  2. If the condition is true execute the statements in the body of the loop and go back to step 1.
  3. If the condition is false drop out of the loop and continue execution with the next statement
The 'while' statement

• This type of statement is called a loop because it “loops” back around to the top

• This type of statement is called a 'while' loop because it continues to execute while the condition is true
Infinite loops

- The body of a loop must change the value of the loop control variable

- If this does not occur then, assuming the condition was initially true, it will never become false

- If the condition is always true we will never stop executing
The 'for' statement

• Here is a simple example of a 'for' loop:

```python
for i in range(10):
    print(i)
```

• This loop prints out numbers from 0 to 9

• There are more complicated versions of the 'for' loop that we will see later
The 'break' statement

- Sometimes we don't know that it is time to end a loop until we are halfway through the body.

- For example, suppose we want to take input from the user until they type 'done'.

The 'break' statement

• We could write:

```python
while True:
    line = input('> ')
    if line == 'done':
        break
    print(line)
print('done')
```
The 'break' statement

• The loop condition is 'True' creating an intentional infinite loop

• The loop will execute until the 'if' statement condition is true

• Then we 'break' out of the loop
The 'break' statement

- The 'break' statement works for both 'while' and 'for' type loops

```python
n = int(input('Enter a number: '))
for i in range(1, n):
    if i > 10:
        print('Sorry, ten is all you get!')
        break
    print(i)
```