Module 4: Repetition Structures

Instructor: Jonny C.H Yu

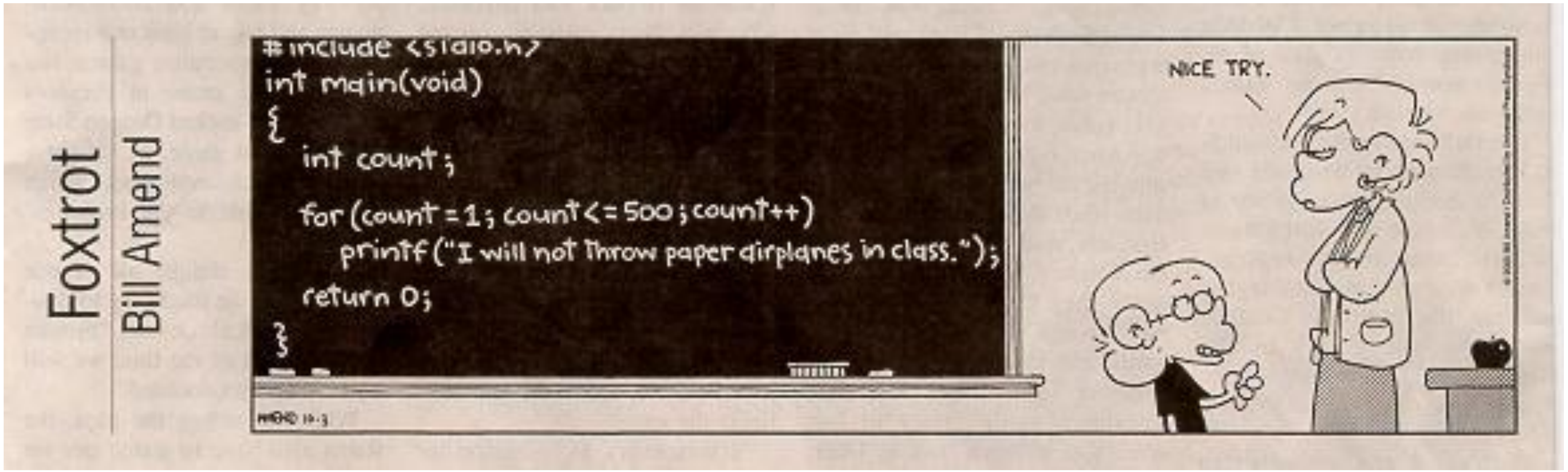
ALL programs can be written using three forms of control:

- Sequential structure (we have learned)
- Decision structure (we have learned)
- Repetition structure (this module)

Bohm, C. and G. Jacopini (1966). “Flow Diagrams, Turning Machines and Languages with Only Two Formation Rules, Communications of ACM, 9(5), 366-371.

Introduction to Repetition Structures

- To execute the same group of statements a few times



```
#include <stdio.h>
int main(void)
{
  int count;
  for (count = 1; count <= 500; count++)
     printf("I will not throw paper airplanes in class.");
  return 0;
}
```

I will not throw paper airplanes in class
I will not throw paper airplanes in class
I will not throw paper airplanes in class
iStockphoto

Introduction to Repetition Structures

- Often must write code that performs the same task multiple times
  - Disadvantages to duplicating code
    - Makes program large
    - Time consuming
    - May need to be corrected in many places
- <u>Repetition structure</u>: makes computer repeat included code as necessary
  - Includes <u>condition-controlled</u> loops and <u>count-controlled</u> loops

Repetition Structure: Condition-Controlled Loop

CONCEPT: A condition-controlled loop causes a statement or set of statements to repeat as long as a condition is true.

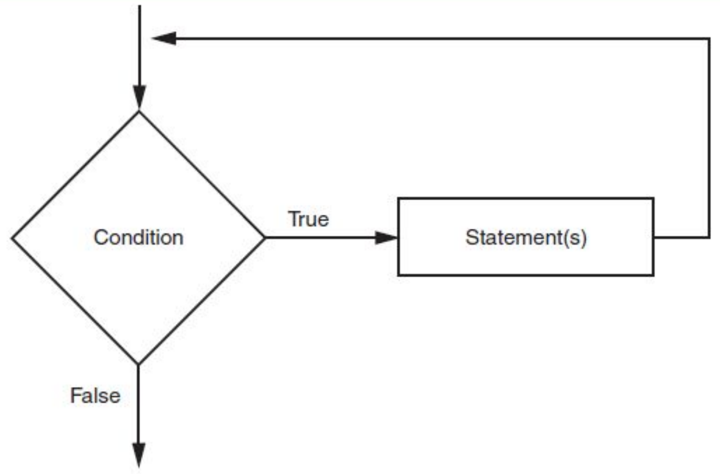
In Python, you use the while statement to write a condition-controlled loop.

The `while` Loop: a Condition-Controlled Loop

- `while` loop: while condition is true, do something
  - Two parts:
    - Condition tested for true or false value
    - Statements repeated as long as condition is true
  - In flow chart, line goes back to previous part
  - General format:

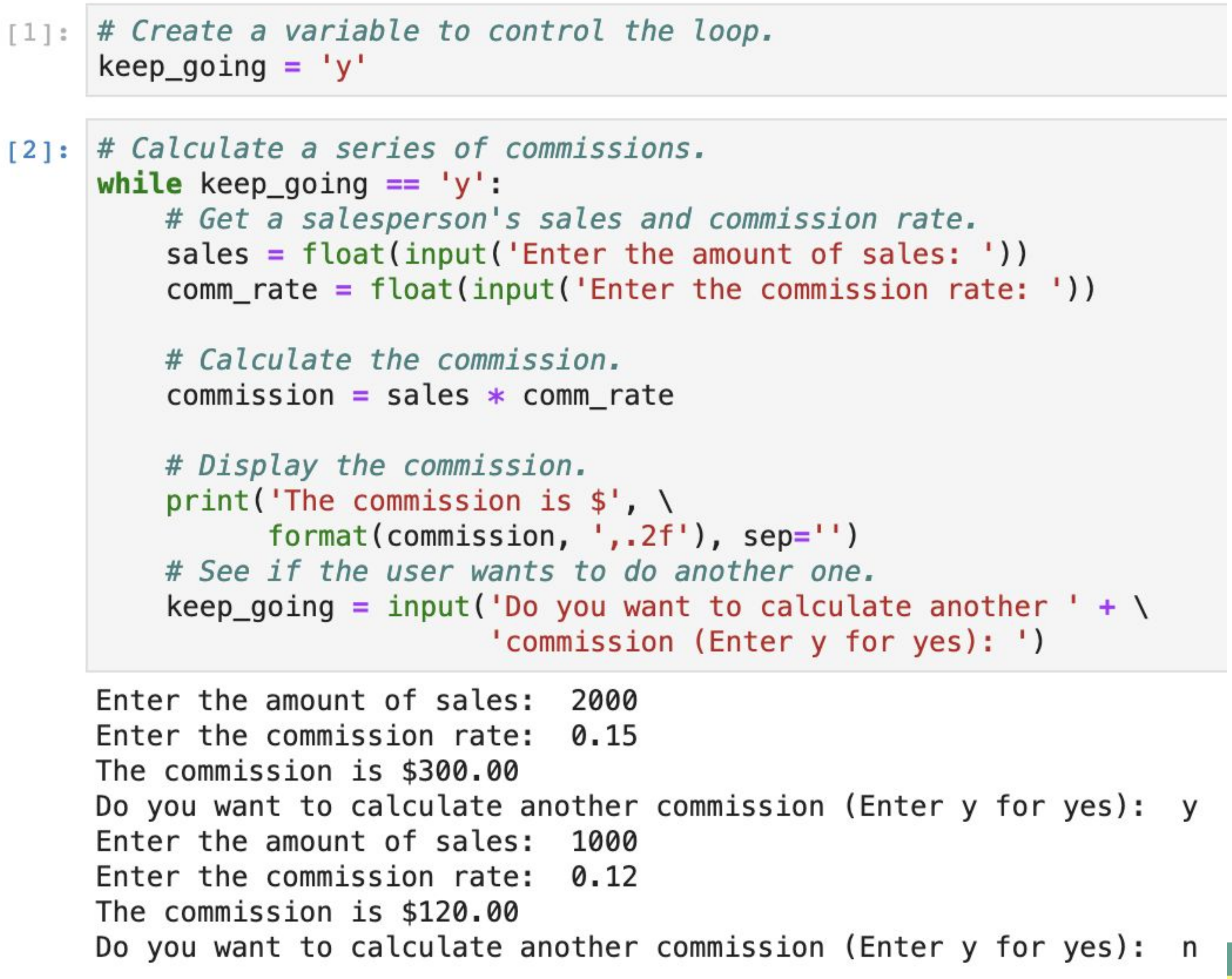
```
while condition:
    statement
    statement
    ...
```

Figure 4-1 The logic of a `while` loop



Example

- Write a program to calculate sales commission for several sales persons.



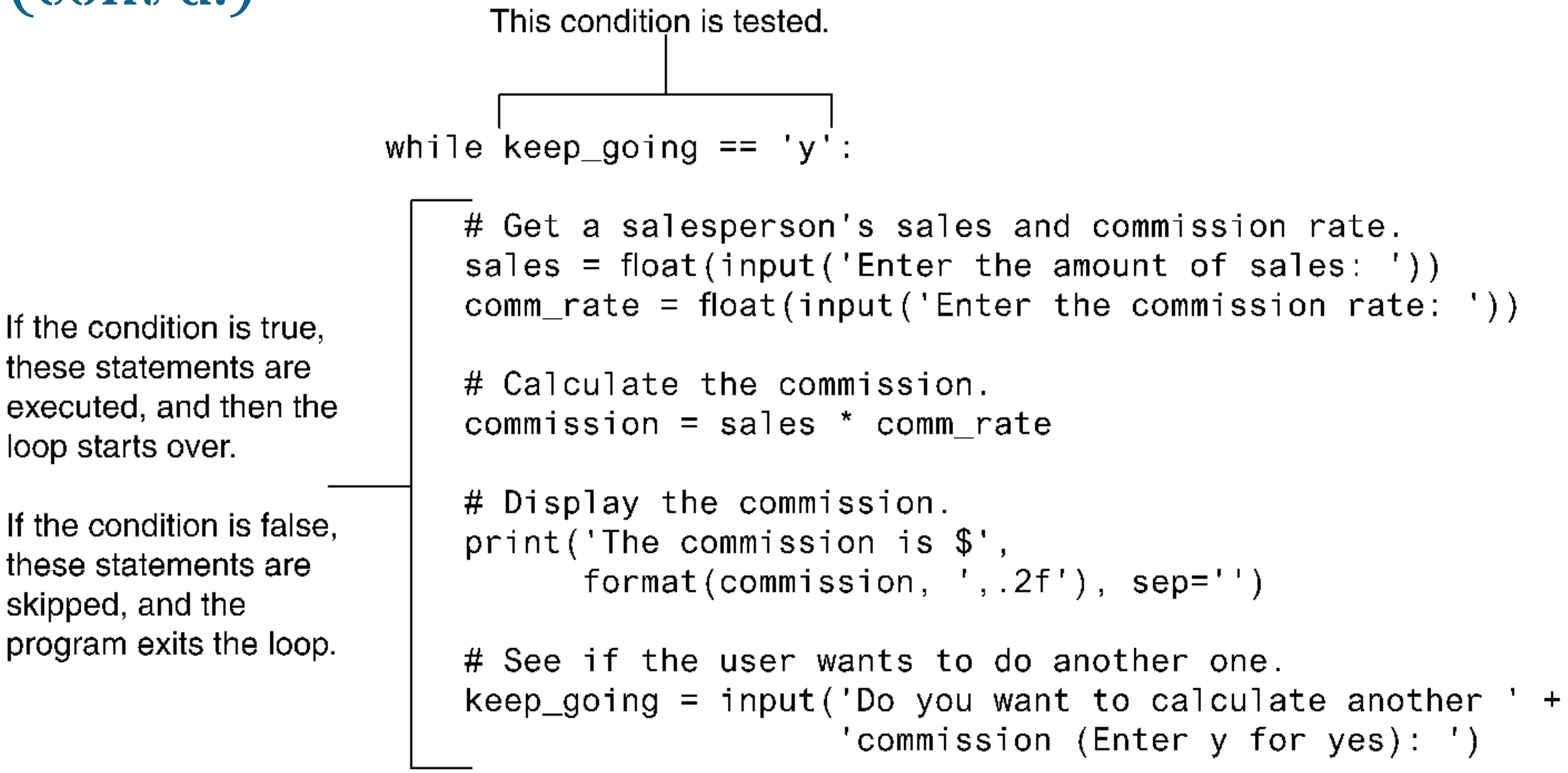
```
[1]: # Create a variable to control the loop.
     keep_going = 'y'
```

```
[2]: # Calculate a series of commissions.
     while keep_going == 'y':
         # Get a salesperson's sales and commission rate.
         sales = float(input('Enter the amount of sales: '))
         comm_rate = float(input('Enter the commission rate: '))

         # Calculate the commission.
         commission = sales * comm_rate

         # Display the commission.
         print('The commission is $', \
               format(commission, ',.2f'), sep='')
         # See if the user wants to do another one.
         keep_going = input('Do you want to calculate another ' + \
                            'commission (Enter y for yes): ')
```

```
Enter the amount of sales:  2000
Enter the commission rate:  0.15
The commission is $300.00
Do you want to calculate another commission (Enter y for yes):  y
Enter the amount of sales:  1000
Enter the commission rate:  0.12
The commission is $120.00
Do you want to calculate another commission (Enter y for yes):  n
```

Let us try it! Download Codes_Module04.zip, upzip and run M4_SalesCommission.ipynb

The `while` Loop: a Condition-Controlled Loop (cont'd.)



This condition is tested.

```
while keep_going == 'y':

    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))

    # Calculate the commission.
    commission = sales * comm_rate

    # Display the commission.
    print('The commission is $',
          format(commission, ',.2f'), sep='')

    # See if the user wants to do another one.
    keep_going = input('Do you want to calculate another ' +
                       'commission (Enter y for yes): ')
```

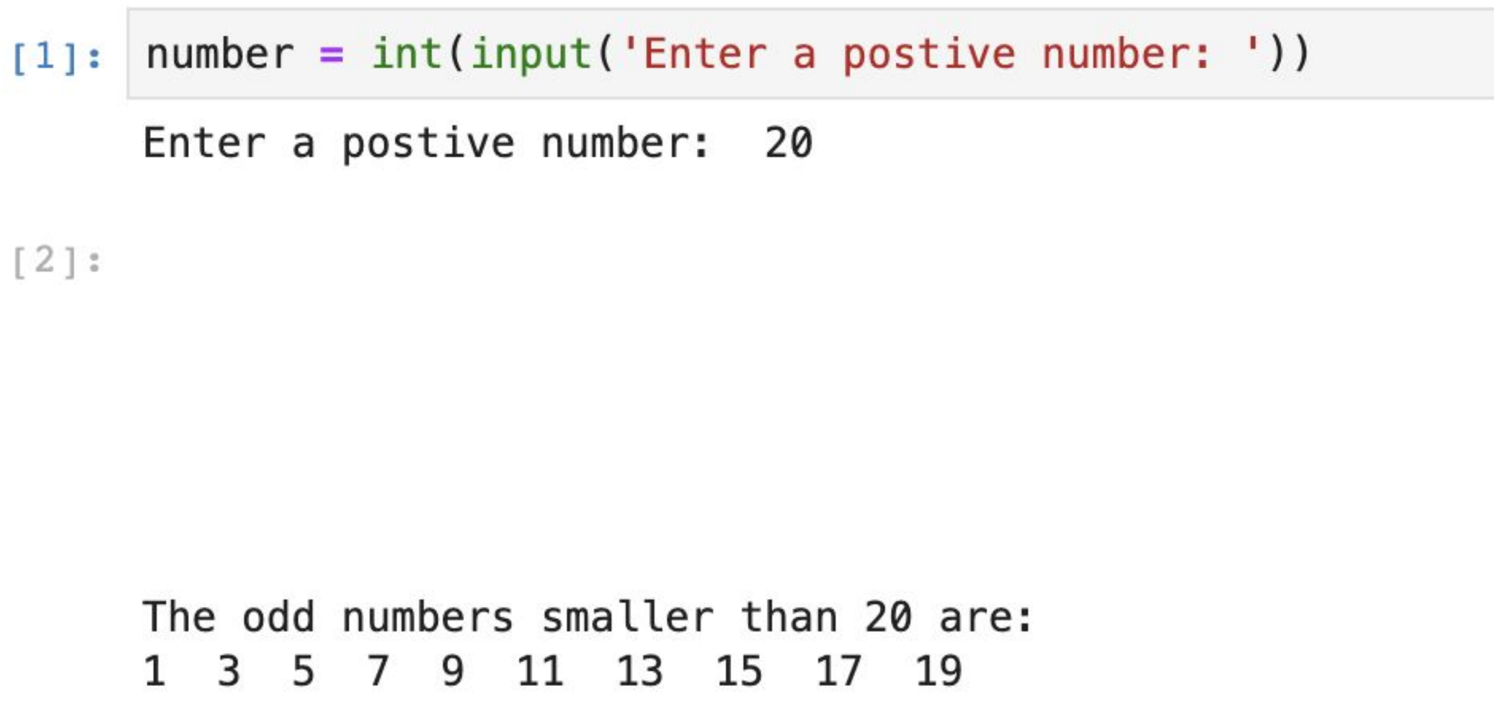
If the condition is true, these statements are executed, and then the loop starts over.

If the condition is false, these statements are skipped, and the program exits the loop.

The `while` Loop: a Condition-Controlled Loop (cont'd.)

- In order for a loop to stop executing, something has to happen inside the loop to make the condition false
- Iteration: one execution of the body of a loop
- `while` loop is known as a *pretest* loop
  - Tests condition before performing an iteration
    - Will never execute if condition is false to start with
    - Requires performing some steps prior to the loop

Exercise: Print Odd Numbers

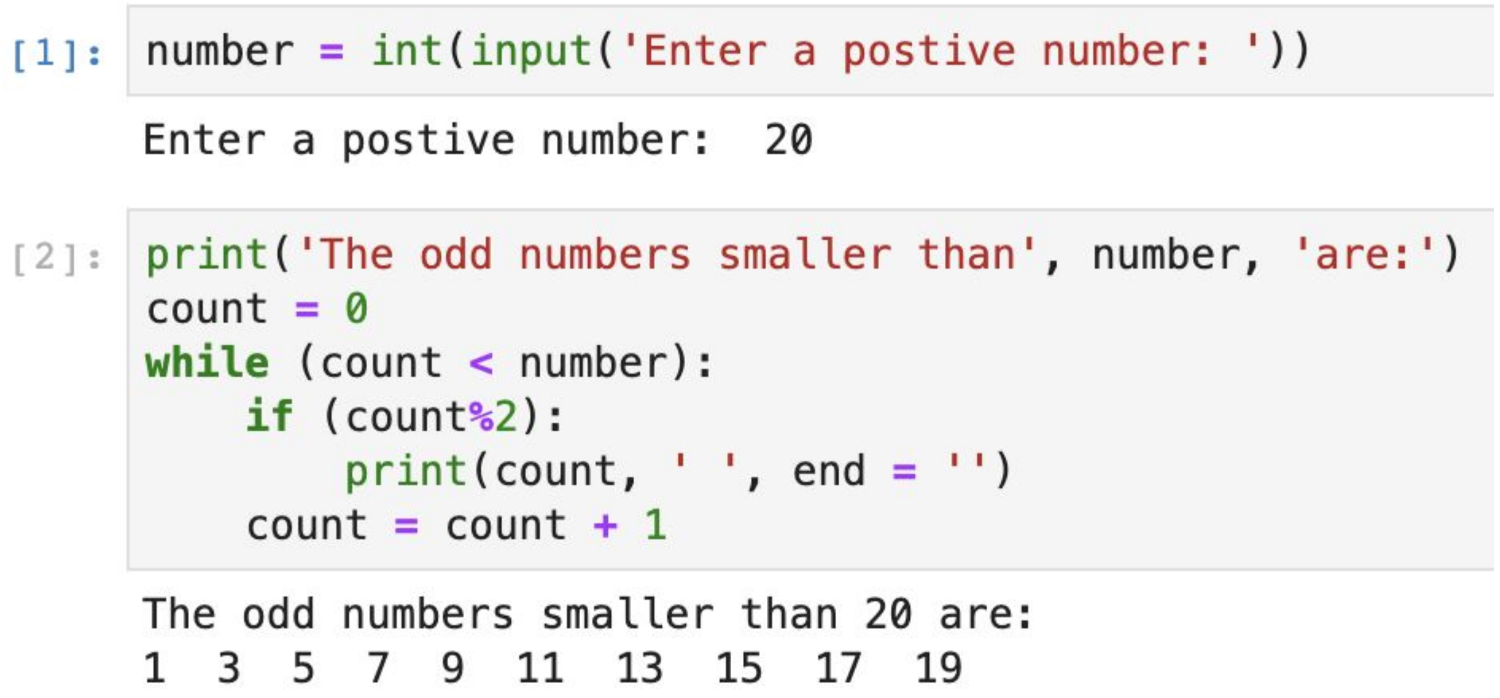


```
[1]: number = int(input('Enter a postive number: '))

     Enter a postive number:  20

[2]:
```

```
The odd numbers smaller than 20 are:
1  3  5  7  9  11  13  15  17  19
```

Exercise: Print Odd Numbers (Ans)

- Ask the user to enter a positive integer and print all the odd numbers smaller than the input integer in order.



```
[1]: number = int(input('Enter a postive number: '))

     Enter a postive number:  20

[2]: print('The odd numbers smaller than', number, 'are:')
     count = 0
     while (count < number):
         if (count%2):
             print(count, ' ', end = '')
         count = count + 1

     The odd numbers smaller than 20 are:
     1   3   5   7   9   11   13   15   17   19
```

The Augmented Assignment Operators

- In many assignment statements, the variable on the left side of the = operator also appears on the right side of the = operator
- Augmented assignment operators: special set of operators designed for this type of job
  - Shorthand operators

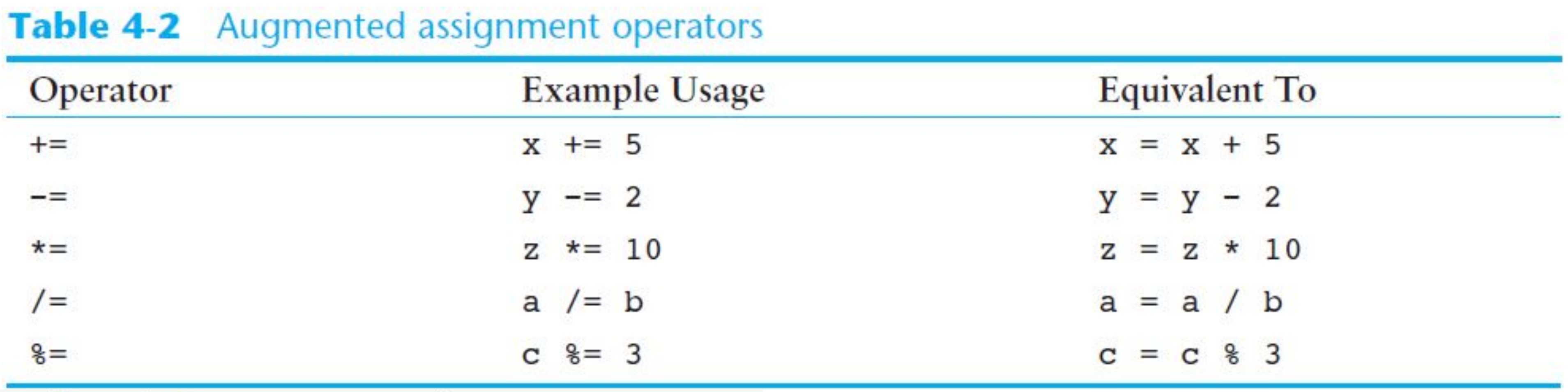
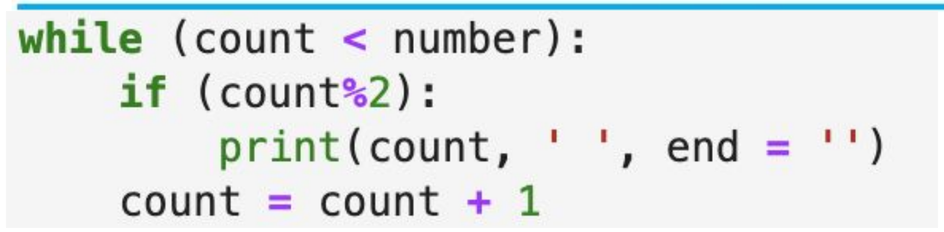


Table 4-2 Augmented assignment operators

Operator	Example Usage	Equivalent To
`+=`	`x += 5`	`x = x + 5`
`-=`	`y -= 2`	`y = y - 2`
`*=`	`z *= 10`	`z = z * 10`
`/=`	`a /= b`	`a = a / b`
`%=`	`c %= 3`	`c = c % 3`



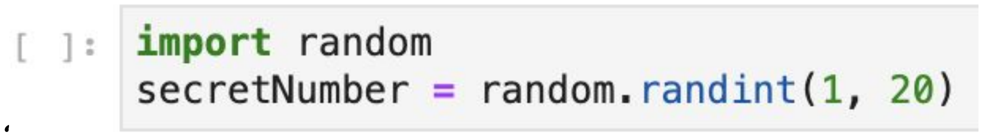
```python
while (count < number):
    if (count%2):
        print(count, ' ', end = '')
    count = count + 1
```

Fun Exercise: Guess the Number

- Write a "Guess the Number" game. The computer will think of a random number from 1 to 20, and ask you to guess it. The computer will tell you if each guess is too high or too low. You win if you can guess the number within six tries.

Program Output (with input shown in **bold**)

```
Hello! Please enter your name: Albert
Well Albert, I am thinking of a number between 1
Take a guess: 1
Your guess is too low.
Take a guess: 2
Your guess is too low.
Take a guess: 3
Your guess is too low.
Take a guess: 4
Your guess is too low.
Take a guess: 5
Your guess is too low.
Take a guess: 6
Your guess is too low. Nope. The number I was thinking
of was 8
```



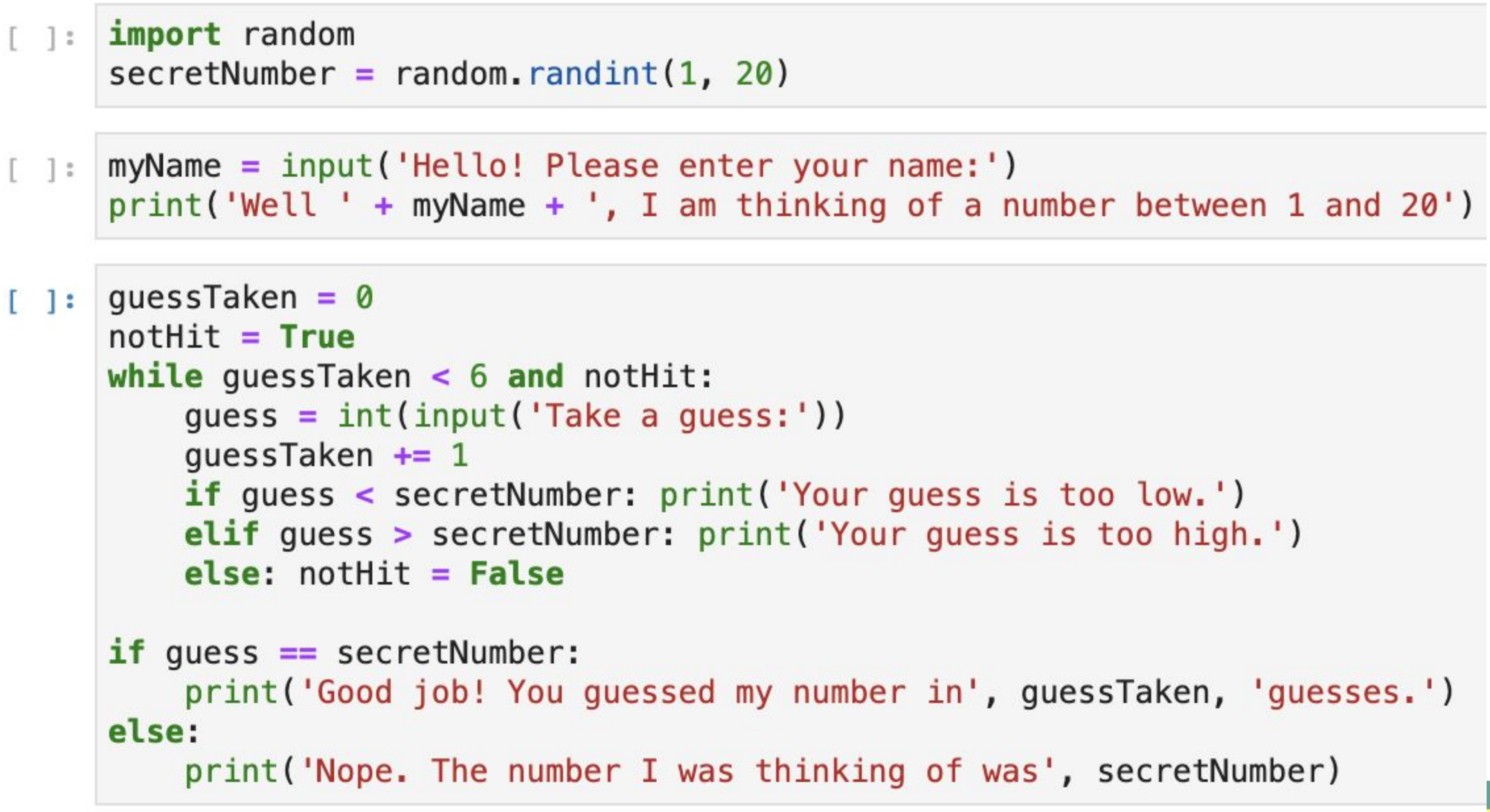
```
[ ]: import random
     secretNumber = random.randint(1, 20)
```

Program Output (with input shown in **bold**)

```
Hello! Please enter your name: David
Well David, I am thinking of a number between 1 and 20
Take a guess: 5
Your guess is too low.
Take a guess: 6
Your guess is too low.
Take a guess: 7
Good job! You guessed my number in 3 guesses.
```

Fun Exercise: Guess the Number (Ans)

- Write a “Guess the Number” game. The computer will think of a random number from 1 to 20, and ask you to guess it. The computer will tell you if each guess is too high or too low. You win if you can guess the number within six tries.



```
[ ]: import random
     secretNumber = random.randint(1, 20)
```

```
[ ]: myName = input('Hello! Please enter your name:')
     print('Well ' + myName + ', I am thinking of a number between 1 and 20')
```

```
[ ]: guessTaken = 0
     notHit = True
     while guessTaken < 6 and notHit:
         guess = int(input('Take a guess:'))
         guessTaken += 1
         if guess < secretNumber: print('Your guess is too low.')
         elif guess > secretNumber: print('Your guess is too high.')
         else: notHit = False

     if guess == secretNumber:
         print('Good job! You guessed my number in', guessTaken, 'guesses.')
     else:
         print('Nope. The number I was thinking of was', secretNumber)
```

Repetition Structure: Count-Controlled Loop

CONCEPT: A count-controlled loop iterates a specific number of times.

In Python, you use the `for` statement to write a count-controlled loop.

The `for` Loop: a Count-Controlled Loop

- Count-Controlled loop: iterates a specific number of times

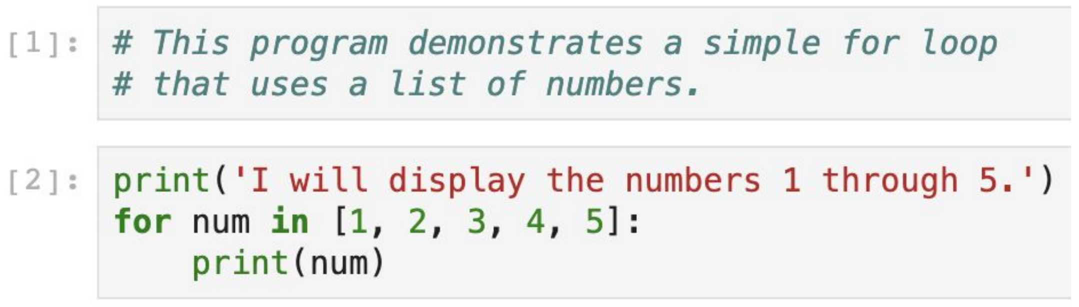
  Use a `for` statement to write count-controlled loop

  - Designed to work with sequence of data items
    - Iterates once for each item in the sequence
  - General format:

```
for variable in [val1, val2, etc]:
  statements
```

  - Target variable: the variable which is the target of the assignment at the beginning of each iteration

Example

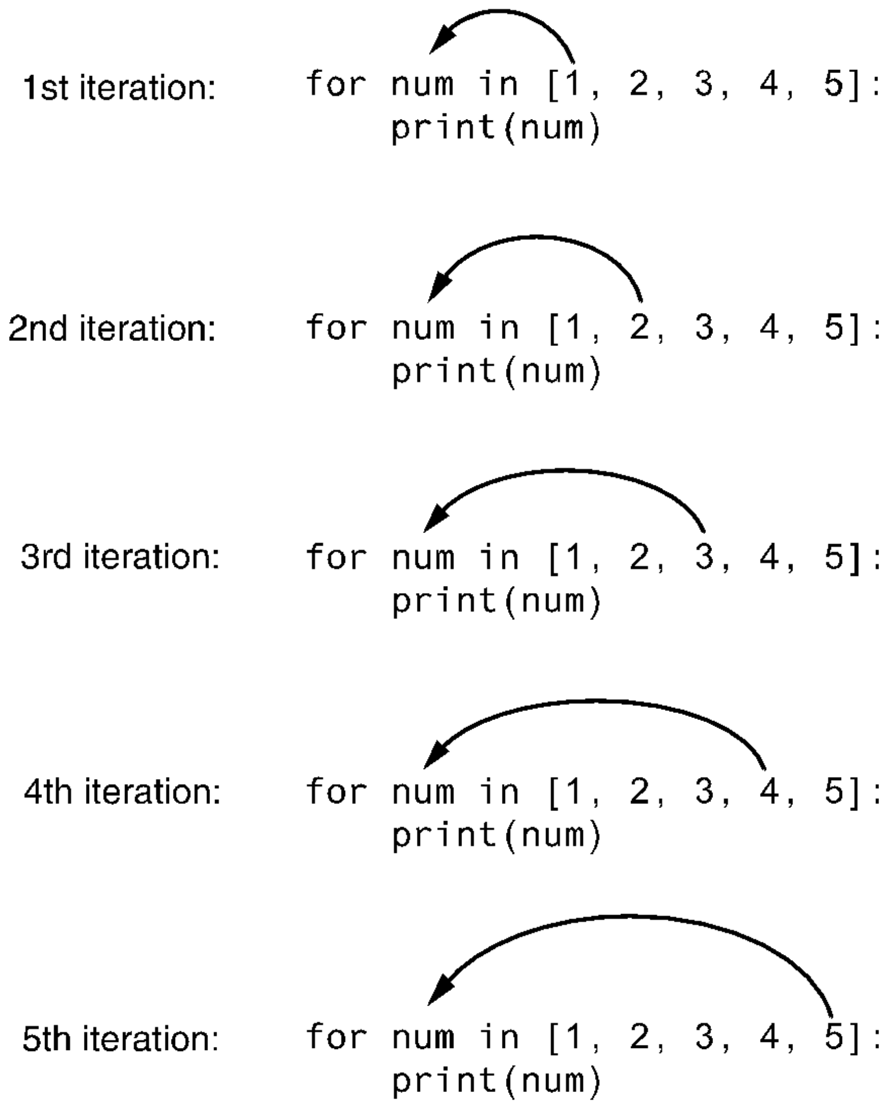


```
[1]: # This program demonstrates a simple for loop
     # that uses a list of numbers.
```

```
[2]: print('I will display the numbers 1 through 5.')
     for num in [1, 2, 3, 4, 5]:
         print(num)
```

Q: What are the outputs?

A:



1st iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

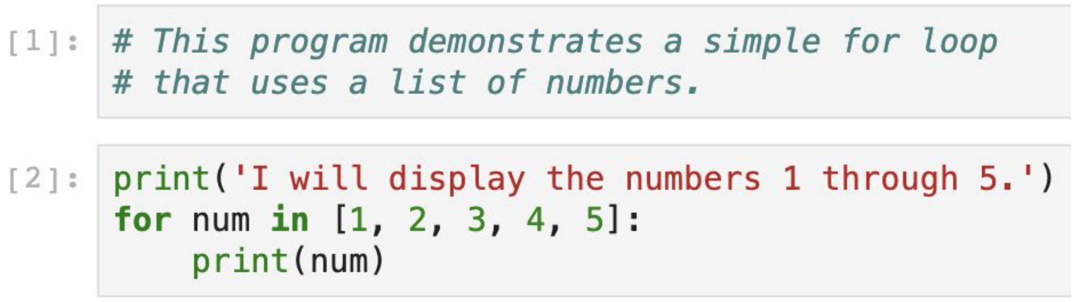
2nd iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

3rd iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

4th iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

5th iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

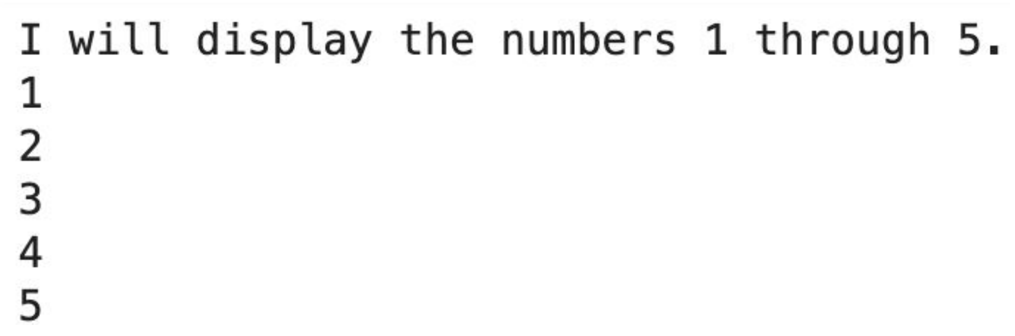
Example (Ans)



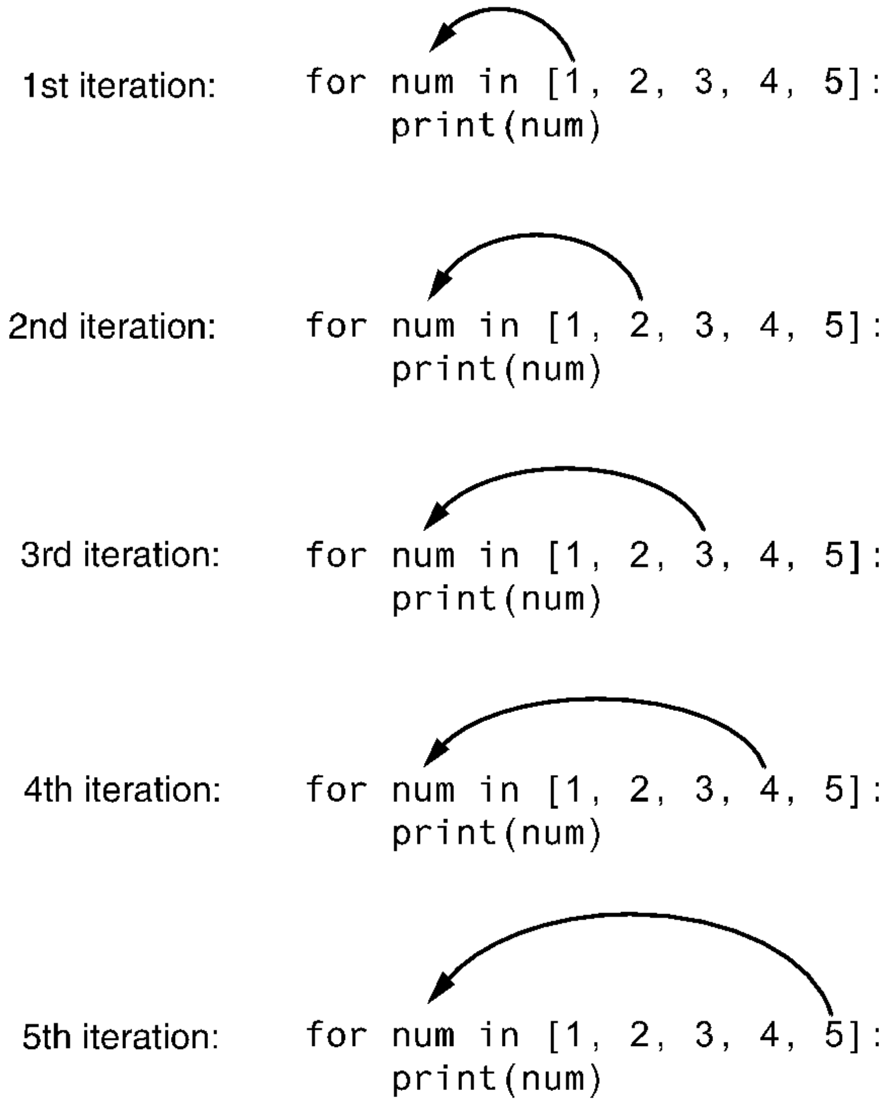
```
[1]: # This program demonstrates a simple for loop
     # that uses a list of numbers.

[2]: print('I will display the numbers 1 through 5.')
     for num in [1, 2, 3, 4, 5]:
         print(num)
```

Q: What are the outputs?

A:



```
I will display the numbers 1 through 5.
1
2
3
4
5
```



1st iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

2nd iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

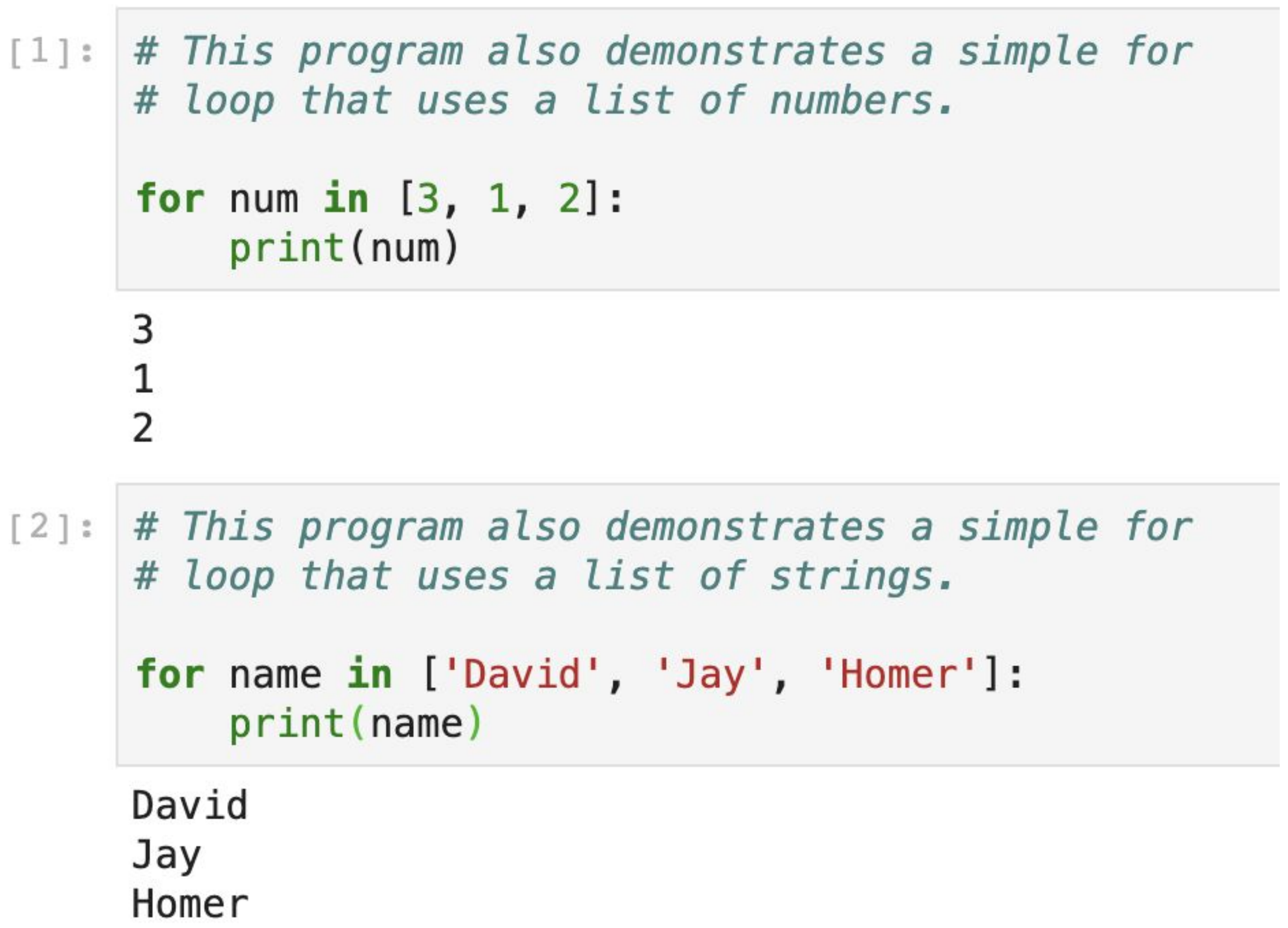
3rd iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

4th iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

5th iteration:
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

M4_SimpleForLoop.ipynb

More Examples

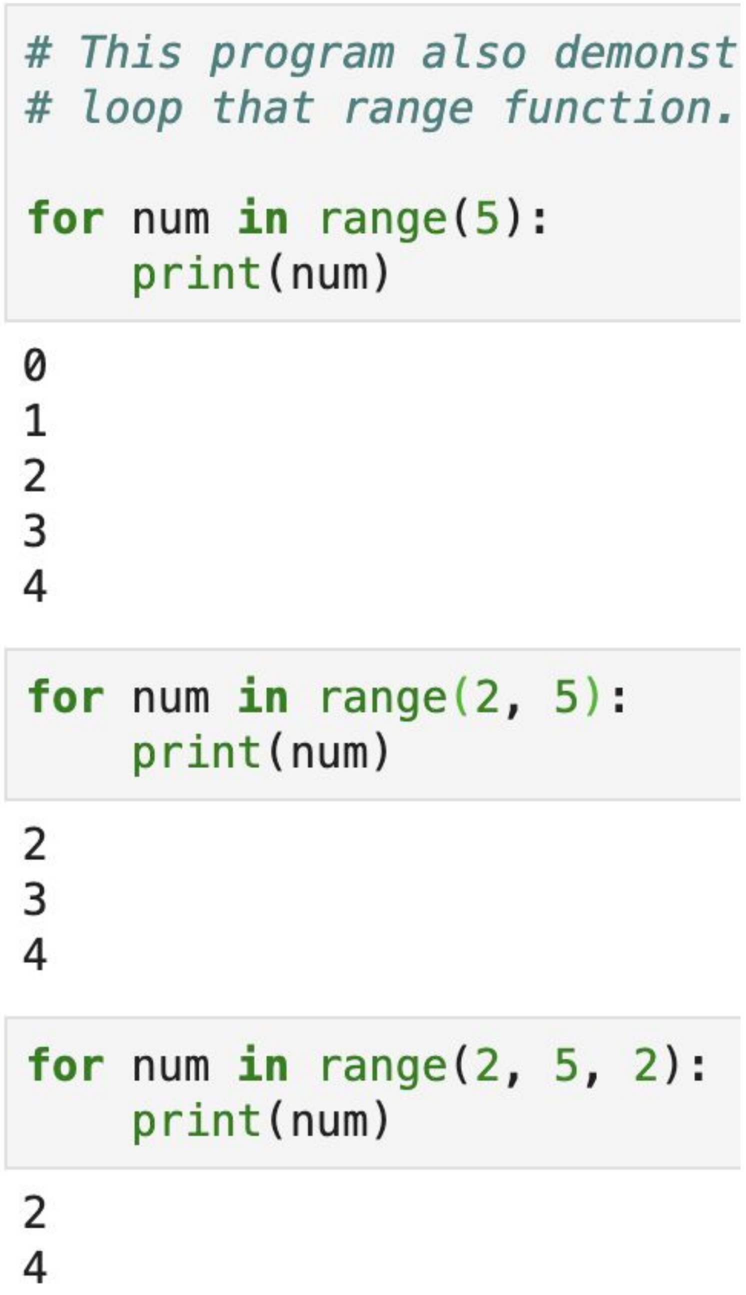


```
[1]: # This program also demonstrates a simple for
     # loop that uses a list of numbers.

     for num in [3, 1, 2]:
         print(num)

     3
     1
     2
```

```
[2]: # This program also demonstrates a simple for
     # loop that uses a list of strings.

     for name in ['David', 'Jay', 'Homer']:
         print(name)

     David
     Jay
     Homer
```

Using the range Function with the for Loop

- The range function simplifies the prc loop
  - range returns an iterable object
    - Iterable: contains a sequence of iterated over
- range characteristics:
  - One argument: used as ending limi
  - Two arguments: starting value and
  - Three arguments: third argument i



```
# This program also demonst
# loop that range function.

for num in range(5):
    print(num)
```

```
0
1
2
3
4
```

```
for num in range(2, 5):
    print(num)
```

```
2
3
4
```

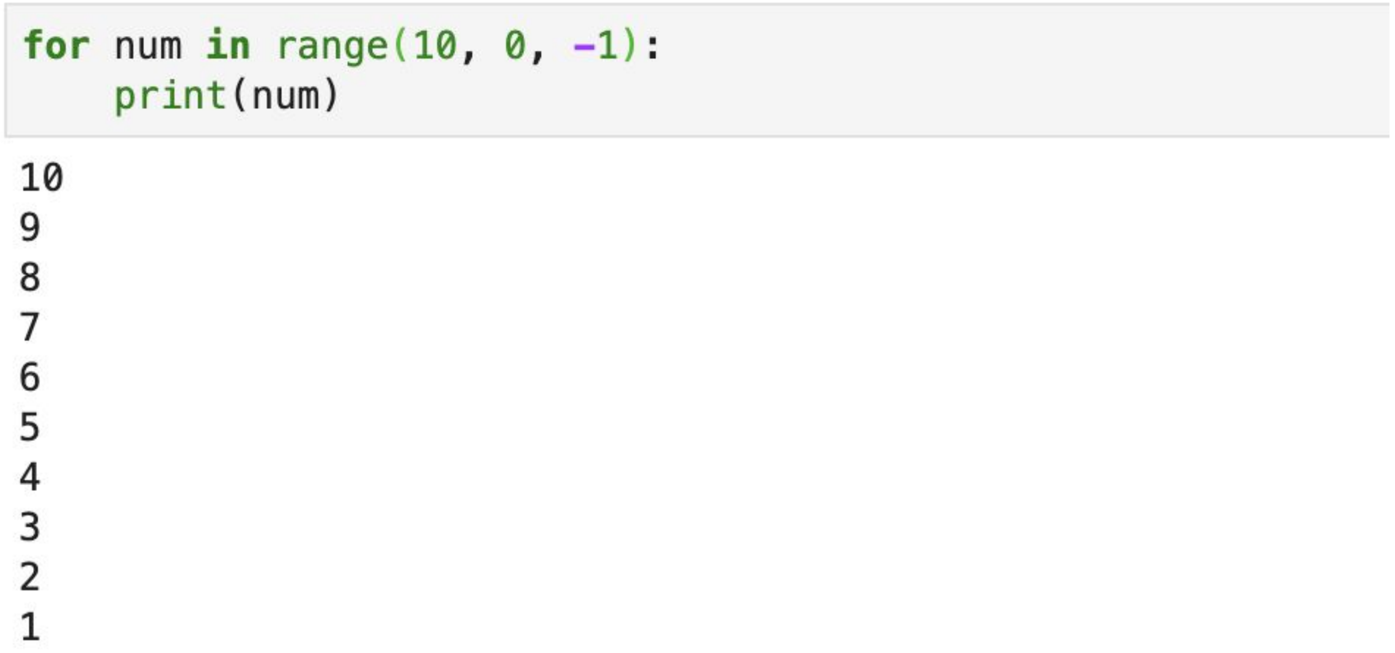
```
for num in range(2, 5, 2):
    print(num)
```

```
2
4
```

M4_SimpleForLoop2.ipynb

Generating an Iterable Sequence that Ranges from Highest to Lowest

- The `range` function can be used to generate a sequence with numbers in descending order
  - Make sure starting number is larger than end limit, and step value is negative
  - Example: `range (10, 0, -1)`

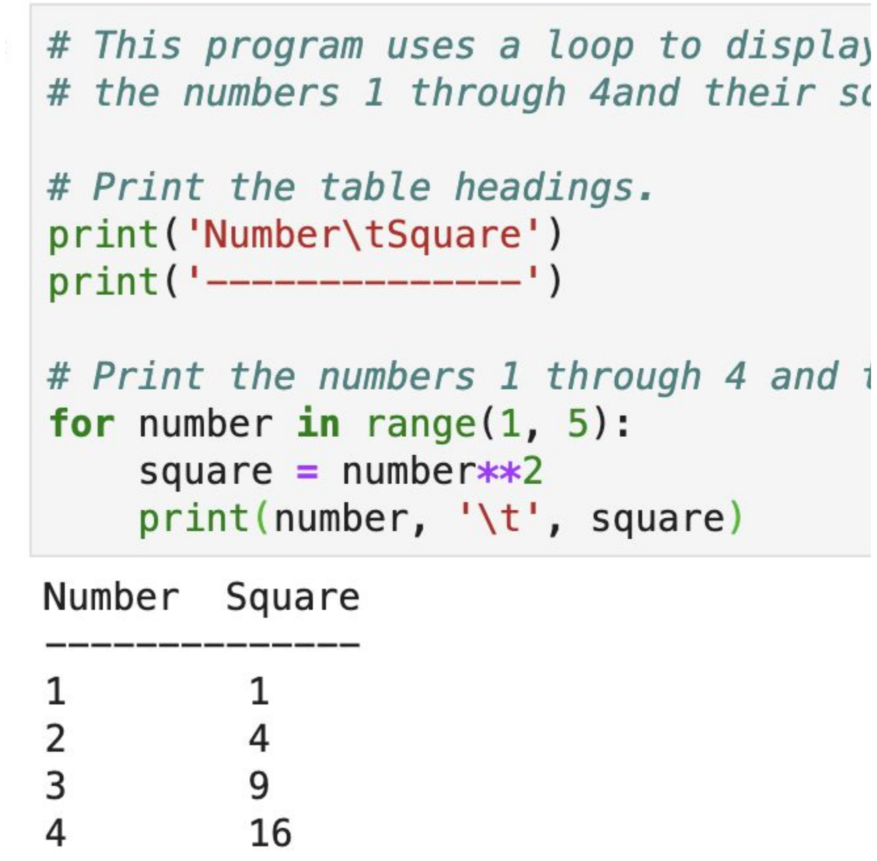


```
for num in range(10, 0, -1):
    print(num)
```

```
10
9
8
7
6
5
4
3
2
1
```

Example: Using the Target Variable Inside the Loop

- Purpose of target variable is to reference each item in a sequence as the loop iterates
- Target variable can be used in calculations or tasks in the body of the loop
  - Example: calculate square root of each number in a range



```
# This program uses a loop to displa
# the numbers 1 through 4and their s

# Print the table headings.
print('Number\tSquare')
print('--------------')

# Print the numbers 1 through 4 and
for number in range(1, 5):
    square = number**2
    print(number, '\t', square)
```

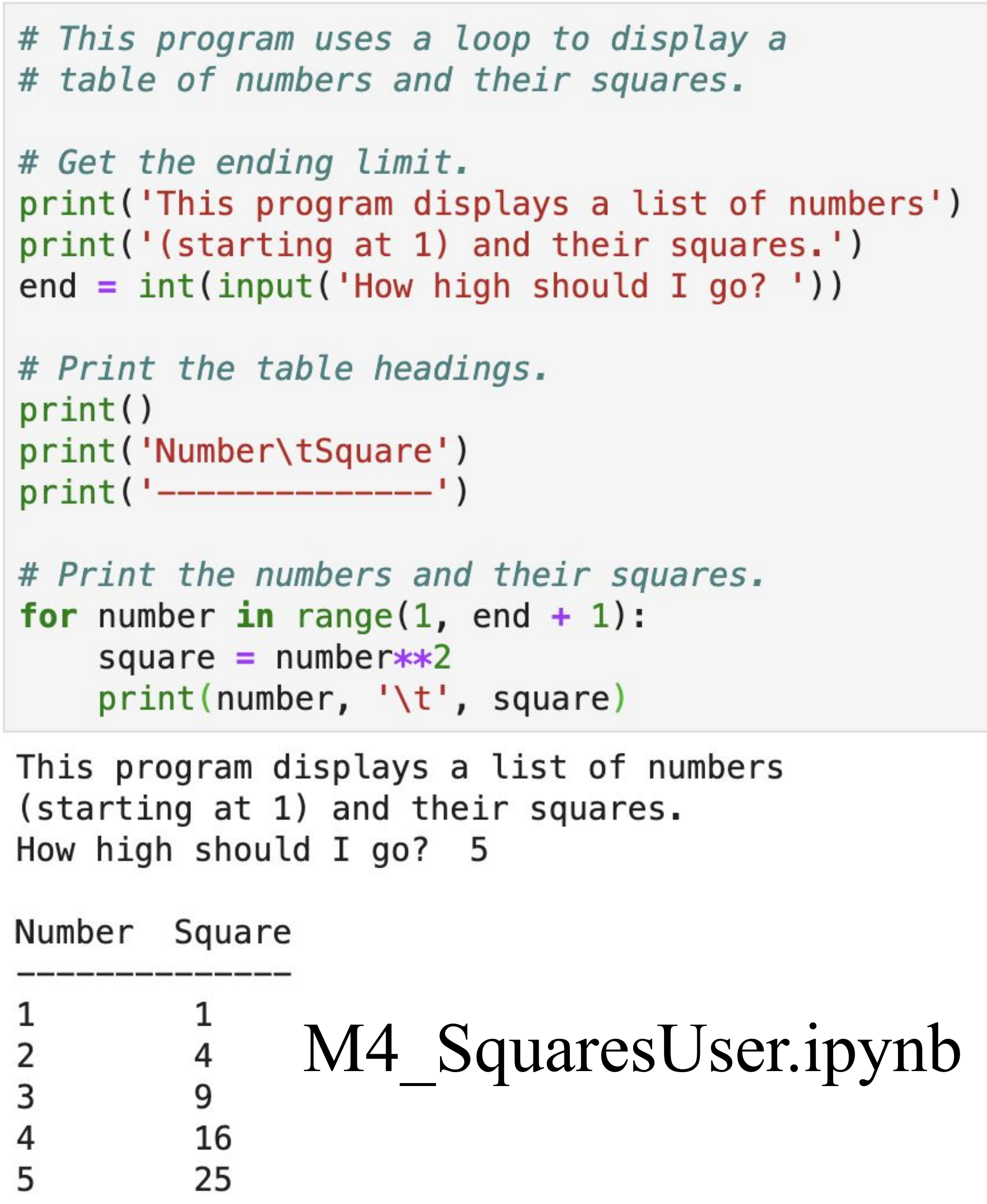
```
Number  Square
--------------
1       1
2       4
3       9
4       16
```

M4_Squares.ipynb

Laboratory for Artificial Intelligence and Multiscale Modeling

Example: Letting the User Control the Loop Iterations

- Sometimes the programmer does not know exactly how many times the loop will execute
- Can receive range inputs from the user, place them in variables, and call the `range` function in the for clause using these variables
    - Be sure to consider the end cases: `range` does not include the ending limit



```python
# This program uses a loop to display a
# table of numbers and their squares.

# Get the ending limit.
print('This program displays a list of numbers')
print('(starting at 1) and their squares.')
end = int(input('How high should I go? '))

# Print the table headings.
print()
print('Number\tSquare')
print('--------------')

# Print the numbers and their squares.
for number in range(1, end + 1):
    square = number**2
    print(number, '\t', square)
```

```
This program displays a list of numbers
(starting at 1) and their squares.
How high should I go?  5

Number  Square
--------------
1        1
2        4
3        9
4        16
5        25
```

M4_SquaresUser.ipynb

Exercise

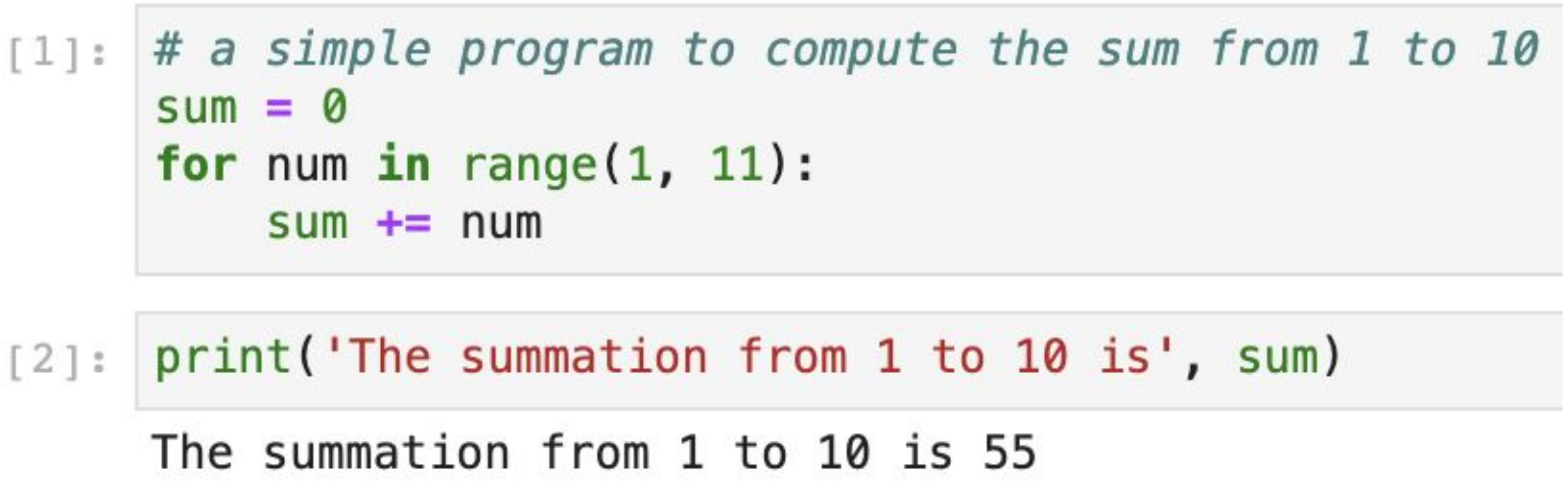
- Write a program to compute the sum from 1 to 10 using for loop.

```
The summation from 1 to 10 is 55
```

Exercise (Answer)

- Write a program to compute the sum from 1 to 10 using for loop.

The summation from 1 to 10 is 55



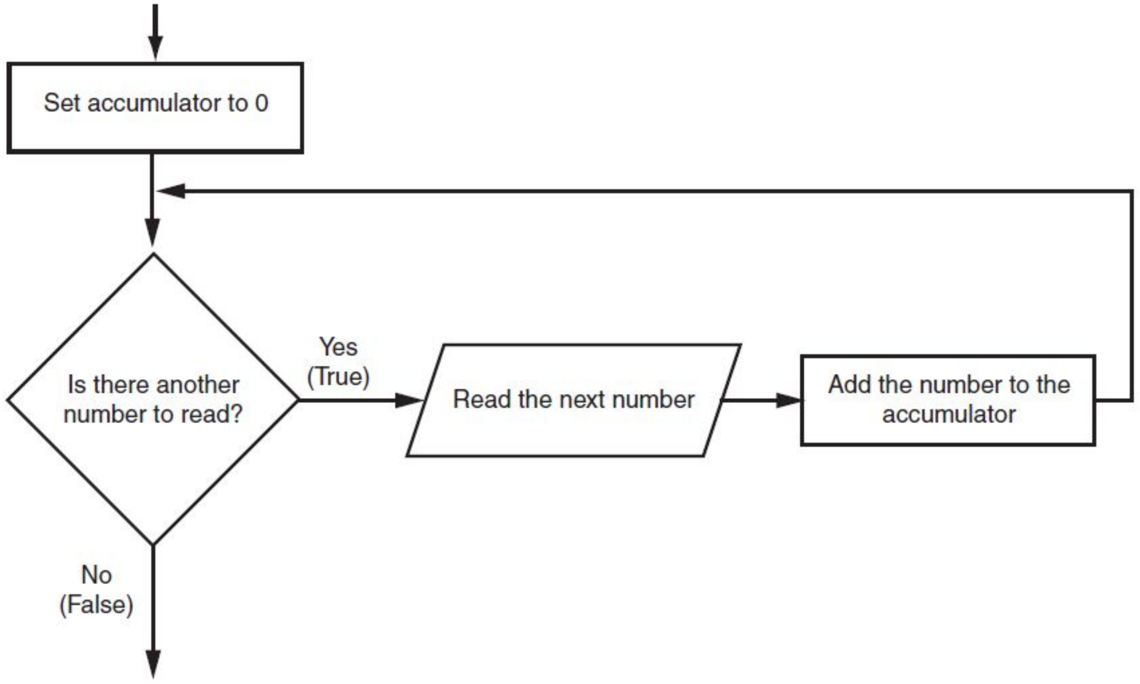
```
[1]: # a simple program to compute the sum from 1 to 10
     sum = 0
     for num in range(1, 11):
         sum += num
```

```
[2]: print('The summation from 1 to 10 is', sum)
```

```
The summation from 1 to 10 is 55
```

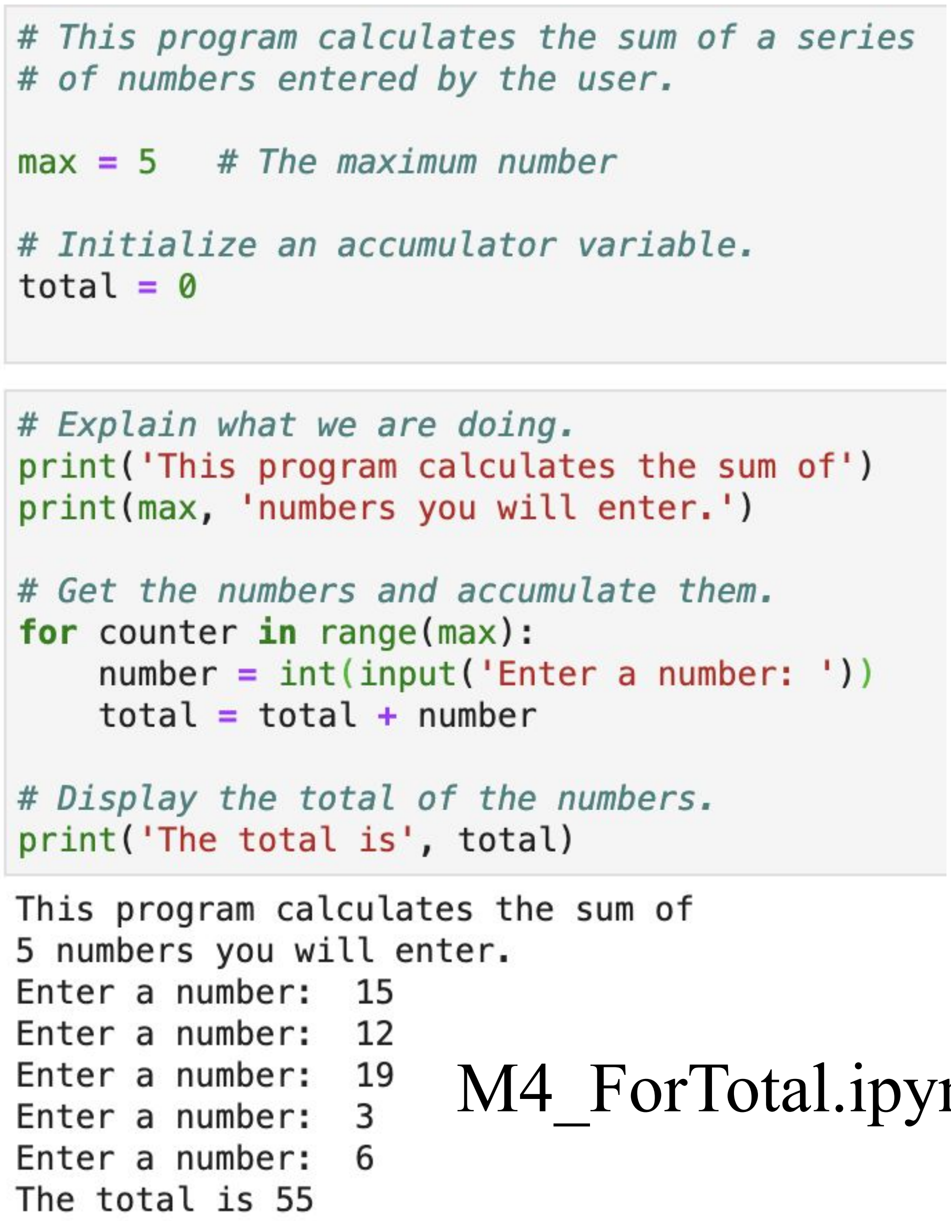
Calculating a Running Total

- Programs often need to calculate a total of a series of numbers
  - Typically include two elements:
    - A loop that reads each number in series
    - An *accumulator* variable
  - Known as program that keeps a running total: accumulates total and reads in series
  - At end of loop, accumulator will reference the total


Set accumulator to 0
Is there another number to read?
Yes (True)
Read the next number
Add the number to the accumulator
No (False)

Calculating a Running Total (Cont')

- Programs often need to calculate a total of a series of numbers
  - Typically include two elements:
    - A loop that reads each number in series
    - An *accumulator* variable
  - Known as program that keeps a running total: accumulates total and reads in series
  - At end of loop, accumulator will reference the total



```
# This program calculates the sum of a series
# of numbers entered by the user.

max = 5    # The maximum number

# Initialize an accumulator variable.
total = 0
```

```
# Explain what we are doing.
print('This program calculates the sum of')
print(max, 'numbers you will enter.')

# Get the numbers and accumulate them.
for counter in range(max):
    number = int(input('Enter a number: '))
    total = total + number

# Display the total of the numbers.
print('The total is', total)
```

```
This program calculates the sum of
5 numbers you will enter.
Enter a number:  15
Enter a number:  12
Enter a number:  19
Enter a number:  3
Enter a number:  6
The total is 55
```

M4_ForTotal.ipynb

Summary

- This module covered:
  - The basics of repetition structures, including:
    - Condition-controlled loops (while loop)
    - Count-controlled loops (for loop)
  - How to initialize, update and control the condition and target in the while loop.
  - How to use range function for the for loop.
  - How to calculate the total of a series of numbers.
  - How to generate random number in Python.