Introduction to: Computers & Programming: Loops in Python

Adam Meyers
New York University

Outline

- What is a Loop?
- While Loops
- For Loops
- Examples
- Nested Loops

What is a Loop?

- Loops are control structures
  - A block of code repeats
  - The extent of the repetition is usually limited in some way
- Two kinds of Loops in Python
  - ***while*** loops
    - The evaluation of a boolean expression determines when the repetition stops
    - Changes in values of variables lead to different evaluations of the boolean expression on each repetition
    - When the expression is evaluated as ***False***, the loop halts
    - If the expression can never evaluate as ***False***, the loop is endless
  - ***for*** loops
    - The length of a sequence determines how many times the body executes
    - The loop uses one member of the sequence at a time, until there are no more members

An Endless Loop

- Example

```
def endless_timer ():
    import time
    now = 0
    while (True):
        time.sleep(1)
        now = now + 1
        print(now)
```

- This loop will keep counting seconds until stopped with a Control-C

What is a ***while*** Loop?

- A while loop consists of:
  - The word ***while***
  - A boolean expression (***True*** on the last slide)
  - A colon :
  - The body: an indented block of instructions
- The body of the loop repeats
  - until the boolean expression is False
- The loop on the previous slide is endless
  - because boolean expression is never False.
  - Any program can be stopped using Control-C

What is a *while* Loop? 2

- A loop that iterates a limited number of times

```
def seconds_stop_watch (total_seconds):
    import time
    now = 0
    while (now <  total_seconds):
        time.sleep(1)
        now = now + 1
        print(now)
```

- If we call ***seconds_stop_watch*** with ***5*** as an argument
  - The variable ***now*** is initialized to 0
  - The loop iterates 5 times
  - Each time: a second passes, 1 is added to now and now is printed
  - In this way, 1 to 5 is printed over 5 seconds
- How many times would a loop beginning ***while (False):*** repeat?

A sample *for* loop

- This function simulates a 60 second timer

```
def one_minute_timer ():
    print(0)
    for second in range(60):
        time.sleep(1)
        print(second + 1)
```

- The function prints 0, then enters a ***for*** loop
  - The loop iterates through a list of numbers from 0 to 59
    - The variable ***second*** is assigned that number as a value
    - The system waits one second
    - The system prints ***second*** + 1

New Material Introduced in the ***one_minute_timer*** function

- The ***range*** function
  - ***range*** takes one or two integers ***m*** and ***n*** as an arguments
  - when ***m*** is left out, it is (by default) set to 0
  - creates a sequence of numbers from ***m*** to ***n-1***
- A ***for*** loop
  - The first line – ***for*** variable ***in*** sequence:
    - ***for*** and ***in*** are keywords
    - variable can be any legal variable name
    - sequence is an ordered set of items
      - Python sequences includes data types like: ***range***, ***list***, ***string***, …
  - The body of the loop repeats once for each item in the sequence
  - On each iteration, the variable is bound to the next item in the sequence

Looping Through a String

- Using a ***for*** loop

```
def for_string_loop (string):
    for letter in string:
        print(letter)
```

  – for-string-loop('Downward')

- Using a ***while*** loop

```
def while_string_loop (string):
    position = 0
    while(position < len(string))
        print(string[position])
        position = 1 + position
```

Lengths and elements of Sequences

- The function ***len*** returns a sequence's length
  - The number of characters – len('Downward')
  - The number of integers in a range – len(range(60))
  - Etc.
- Elements in a range can be identified by their position, beginning with 0 and ending in one less than the length.
  - 'Downward'[0], range(5,10)[0]
  - 'Downward'[7], range(5,10)[4]
  - 'Downward'[8], range(5,10)[5] --- these are errors

for loops vs. *while* loops

- With some code modification, it is always possible to replace a ***for*** loop with a ***while*** loop, but not the other way around
- ***for*** loops are used for situations where you know the number of iterations ahead of time
  - e.g., looping through sequences
- There is no significant efficiency difference
- The difference relates to ease in which humans can read/write code

Example: Drawing an asterisk triangle

- def draw_n_asterisks(n):

```
for current_length in range(n):
    print('*',end='')
```

  – print can take a named argument
    - End='' indicates what to print at the end of the string
      – the character in between the single quotes
    - In this case, nothing
    - The default is a newline character

- def asterisk_triangle(base_size):

```
for current_length in range(1,base_size+1):
    draw_n_asterisks(current_length)
    print()
```

Drawing an asterisk triangle 2

- Nested Loops – a single function

```
def asterisk_triangle2(base_size):
    for current_length in range(1,base_size+1):
        for n in range(current_length):
            print('*',end='')
        print()
```

- Python indicates depth of nesting via indentation
  - Suppose the last line was indented once

Printing a Multiplication table

- def multiplication_table (high_num):

```
    for num1 in range(1, 1+high_num):
        for num2 in range(1, 1+high_num):
            print(num1,'X',num2, '=', num1*num2)
```

- How does this work?

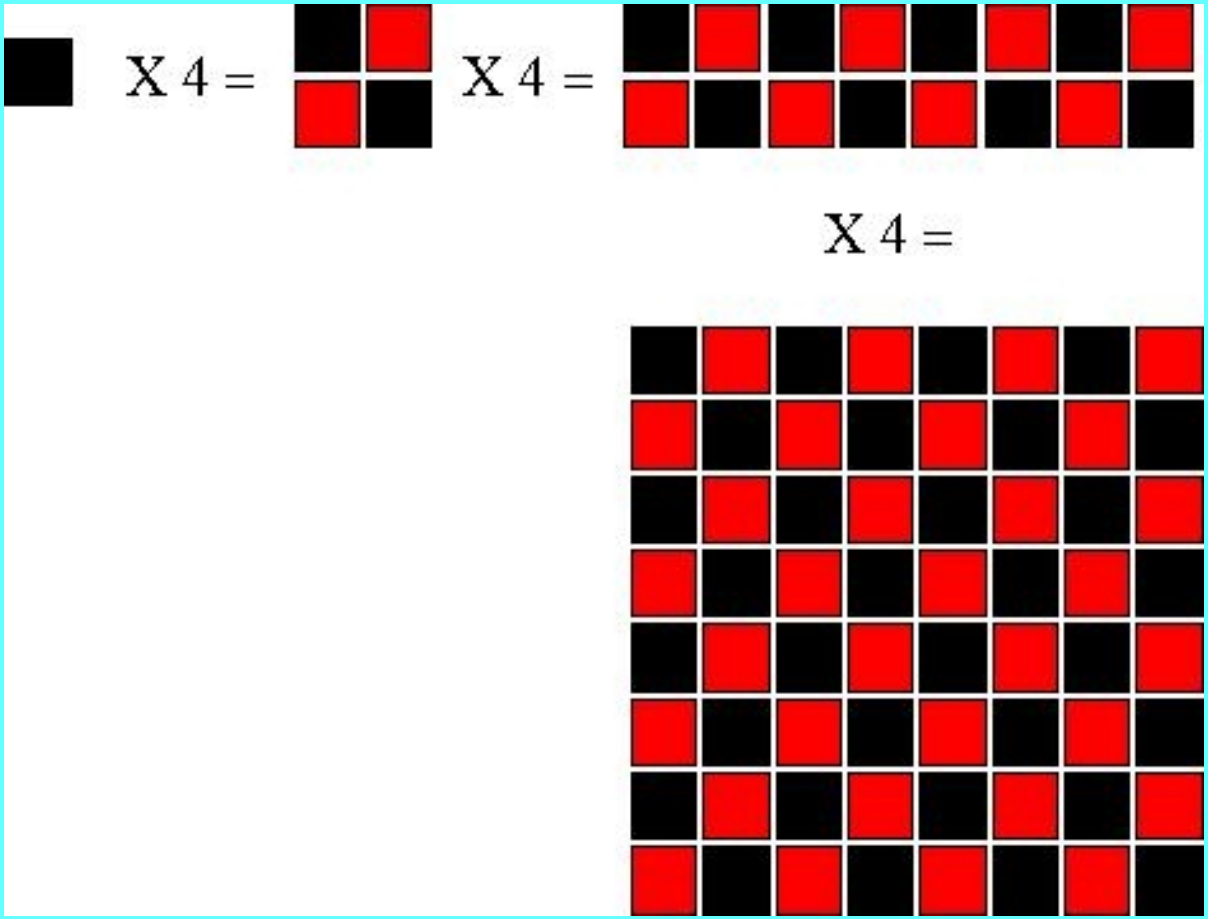
Sample Problem for Class

- Write a function that:
  - Takes three arguments:
    - base_size
    - repetitions
    - hour_glass_or_diamond
  - This function makes a pattern of asterisks that repeats the number of times indicates by repetitions
  - Each cycle consists of two triangles, one the upside down version of each other, both of which have a base of size base_size
  - If hour_glass_or_diamond is in the '*hour glass*' setting, the function draws an upside down triangle and then a right side up triangle
  - If hour_glass_or_diamond is in the '*diamond*' setting, the function draws the right side up triangle first and the upside down one second

One Way to Describe a Checkerboard

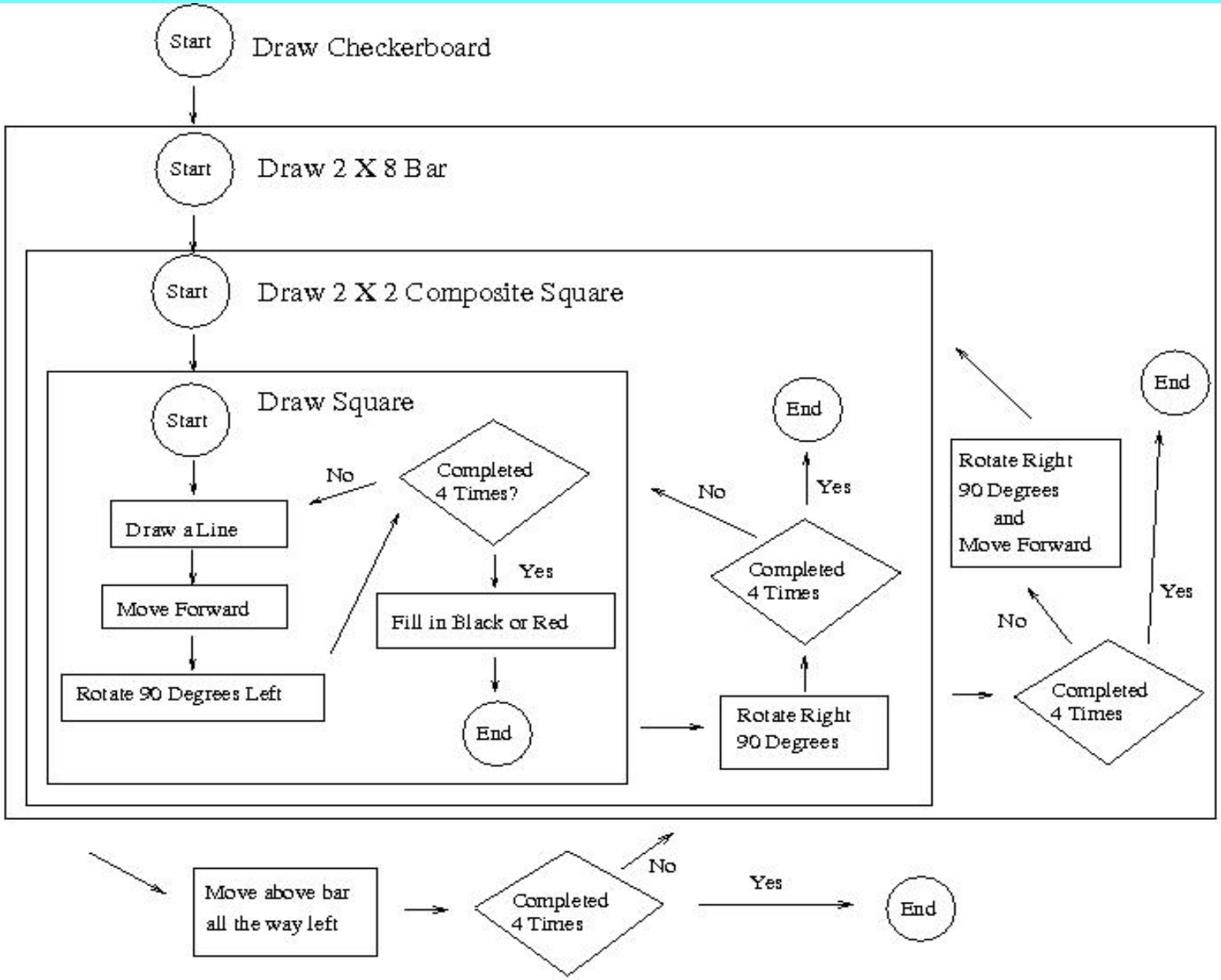
- A Checkerboard is an 8 X 8 square with alternating colors, e.g., red and black
- A Checkerboard can be broken down into 4 bars, each a 2 X 8 bar of alternating colors.
- A 2 X 8 bar of alternating colors can be broken down into 4 composite squares, each consisting of 2 X 2 small squares.

A Pictoral Description



One Way A Turtle Can Draw a Checkerboard?

- The turtle can draw one square and it could fill in with a color of our choice
- The turtle can make four such squares next to each other, forming a composite square.
- It can make four composite squares next to each other, to form a bar.
- It can make four such bars, one under the other to form a checkerboard.


Start
Draw Checkerboard
Start
Draw 2 X 8 Bar
Start
Draw 2 X 2 Composite Square
Start
Draw Square
Draw a Line
Move Forward
Rotate 90 Degrees Left
Completed 4 Times?
No
Yes
Fill in Black or Red
End
Rotate Right 90 Degrees
Completed 4 Times
No
Yes
End
Rotate Right 90 Degrees and Move Forward
Completed 4 Times
No
Yes
End
Move above bar all the way left
Completed 4 Times
No
Yes
End

for_loop_checkerboard.py 1

- Basic setup
  - import turtle
  - my_screen = turtle.Screen()
  - my_screen.setup(0.5,0.75,0,0)
    - width, height, startx, starty
  - my_turtle = turtle.Turtle(shape='turtle')
- draw_colored_turtle_square
- draw_4_black_and_red
- draw_4_black_and_red_4_times
- make_checkerboard

for_loop_checkerboard.py 2

- Setup and then do something 4 times
  - Building block
  - Move in between blocks
- draw_colored_turtle_square
  - Setup: set colors and begin to fill
  - Repeated Steps:
    - put the pen down, move forward, turn left
  - Pick pen up and fill in color
- draw_4_black_and_red
  - Setup: initialize fill color and pen color
  - Repeated Steps: change fill color, draw_colored_turtle_square, turn right, move forward

for_loop_checkerboard.py 3

- draw_4_black_and_red_4_times
  - repeated steps:
    - draw_4_black_and_red
    - move forward
- make_checkerboard
  - Setup: set turtle speed
  - Repeated Steps:
    - draw_4_black_and_red_4_times
    - Turn 180 degrees, move forward, turn 270 degrees, move forward turn 270 degrees

Woops

- While writing the checkerboard program, several mistakes led me to realize some interesting artistic possibilities
  - make_mistake1 – is the result of drawing bars with different angles
- This led me to experiment with different colors, more bars, etc., to make interesting patterns (wacky_bars, wacky_bars2)

About wacky_bars and wacky_bars2

- random.random() – chooses number between 0 and 1
- Colors in the turtle library consist of combinations of red, green and blue
  - Red has a value from 0 to 255
  - Green has a value from 0 to 255
  - Blue has a value from 0 to 255
  - 0,0,0 is black and 255,255,255 is white
- math.ceil(random.random() * 255) – choose a number between 1 and 255
- 3 such random numbers identifies a random color

More about wacky_bars

- Let red, green, blue be 3 numbers between 1 and 255
- Applying the following should produce contrasting colors:
  - red == (red - 128) % 255
  - green = (green - 128) % 255
  - blue = (blue - 128) % 255
- I am unaware of any scientific/artistic significance to this relation, e.g., these pairs of colors are not complementary

Summary

- Loops provide a way to repeat blocks of instructions
- While loops are the most general
  - They require a condition for exiting the loop
    - If the condition is never true, the loop is endless
- For loops provide a simple way of repeating a block
  - once for each element in a sequence
  - or a fixed number of times
- A For loop can always be replaced by an equivalent While loop
- It is often useful to have nested loops (loops within loops)

Homework

- Finish Reading Chapter 4
- Write function that will make a rectangle consisting of any character:
  - Make_character_rectangle(height, width, char)
    - Should make ***height*** rows of ***width*** instances of ***char***
- Write a function that makes a parallelogram of characters:
  - Make_character_parallelogram(height, width, char)
    - The loop should add height-N spaces to the beginning of each line, where N starts out at 1 (the first line) and ends with height (the last line). Thus the parallelogram leans right, but touches the left margin.
- Write a timer that prints out every one tenth of a second
  - It should use the format: Hours:Minutes:Seconds.fraction
  - For example, 00:00:00.0, 00:00:00.1, 00:00:00.2, etc.