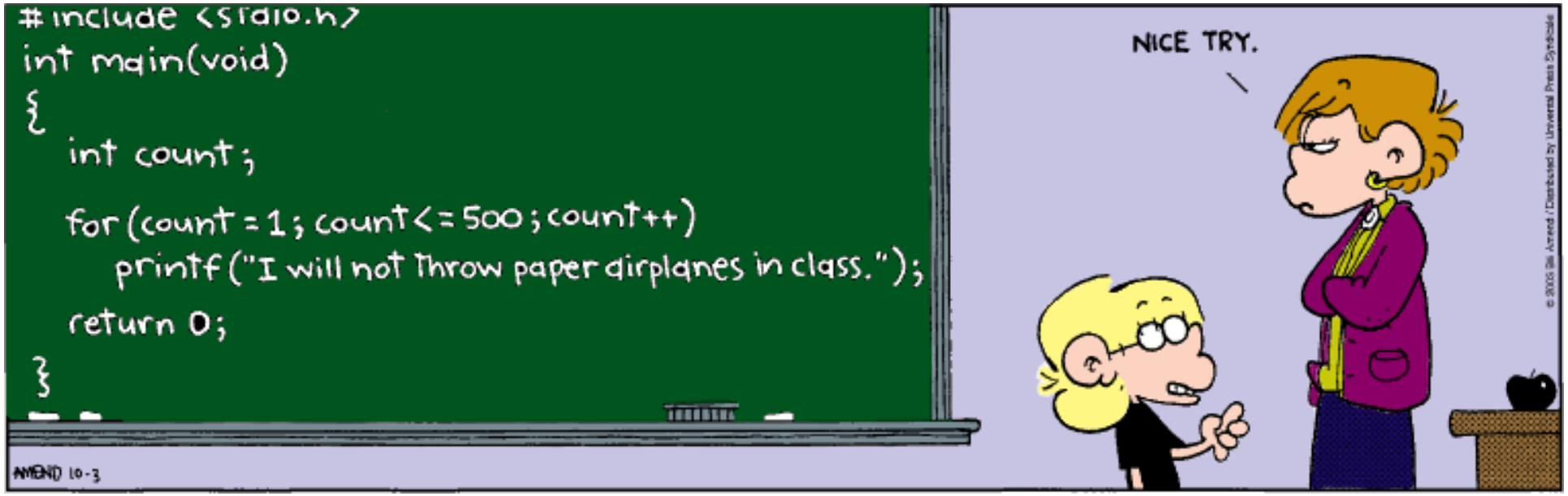
CSc 110, Spring 2017

Lecture 4: Nested Loops and Loop Figures

Adapted from slides by Marty Stepp and Stuart Reges

Can you write this in Python?



Review: for loops

loop: Repeat one or more statements a specified number of times

```
for i in range(1, 6):
    print(i*i)        # square variable i
```

- Output:

```
1
4
9
16
25
```

- The loop repeats 5 times.

Review: print conventions

- `print('', end='')`
- Adding `,end=''` allows you to print without moving to the next line
  - allows you to print on the same line; no advancing to the next line
  - the quotes contain any valid string

```
for i in range(-3, 3):
    print(i, end=' ')
```

  - Output:

```
-3  -2  -1  0  1  2
```

New: Changing step size

- Add a third number to the end of range, this is the step size
  - A negative number will count down instead of up

```
for i in range(10, 0, -1):
    print(i, end=' ')
```

  - output:

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

  - How would we produce the following Rocket Countdown?

```
T-minus 10! 9! 8! 7! 6! 5! 4! 3! 2! 1! blastoff!!
The end.
```

Rocket Countdown

- Use a negative number for step size
- Use `str()` and concatenation in `print()`

```
print("T-minus ")
for i in range(10, 0, -1):
     print(str(i) + "! ", end='')
print("blastoff!!")
print("The end.")
```

- Output:

```
T-minus 10! 9! 8! 7! 6! 5! 4! 3! 2! 1! blastoff!!
The end.
```

Nested loops

- **nested loop**: A loop placed inside another loop.

```
for i in range(1, 6):
    for j in range(1, 11):
        print("*", end="")
    print()         # to end the line
```

- Output:

```
**********
**********
**********
**********
**********
```

- The outer loop repeats 5 times; the inner one 10 times.
  - "sets and reps" exercise analogy

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for i in range(1, 6):
    for j in range(1, i + 1):
        print("*", end='')
    print()
```

- Output:

```
*
**
***
****
*****
```

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for i in range(1, 6):
    for j in range(1, i + 1):
        print(i, end='')
    print()
```

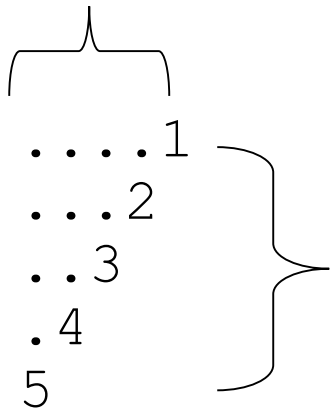
- Output:

```
1
22
333
4444
55555
```

Complex lines

- What nested `for` loops produce the following output?

inner loop (repeated characters on each line)



outer loop (loops 5 times because there are 5 lines)

- We must build multiple complex lines of output using:
  - an *outer "vertical" loop* for each of the lines
  - *inner "horizontal" loop(s)* for the patterns within each line

Outer and inner loop

- First write the outer loop, from 1 to the number of lines.

```
for line in range(1, 6):
    …
```

- Now look at the line contents. Each line has a pattern:
  - Zero or more dots, then a number

```
....1
...2
..3
.4
5
```

  - Observation: the number of dots is related to the line number.

Mapping loops to numbers

```
for count in range(1, 6):
    print( … )
```

- What statement in the body would cause the loop to print:

  4 7 10 13 16

```
for count in range(1, 6):
    print(3 * count + 1, end=' ');
```

Loop tables

```
for count in range(1, 6):
        print(…)
```

- What statement in the body would cause the loop to print:

  `2 7 12 17 22`

- To see patterns, make a table of `count` and the numbers.
  - Each time count goes up by 1, the number should go up by 5.
  - But `count * 5` is too great by 3, so we subtract 3.

`count`	number to print	`5 * count`	`5 * count - 3`
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

Loop tables question

- What statement in the body would cause the loop to print:

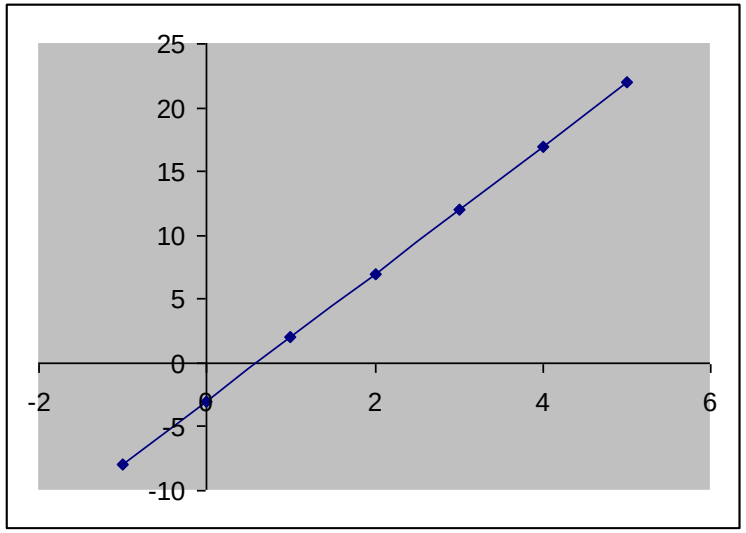
```
17 13 9 5 1
```

- Let's create the loop table together.
  - Each time `count` goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

`count`	number to print	`-4 * count`	`-4 * count + 21`
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

Another view: Slope-intercept

- The next three slides present the mathematical basis for the loop tables. Feel free to skip it.



`count (x)`	number to print (`y`)
1	2
2	7
3	12
4	17
5	22

Another view: Slope-intercept

- *Caution*: This is algebra, not assignment!
- Recall: slope-intercept form (`y = mx + b`)
- Slope is defined as "rise over run" (i.e., rise / run). Since the "run" is always `1` (we increment along `x` by 1), we just need to look at the "rise". The rise is the difference between the `y` values. Thus, the slope (`m`) is the difference between `y` values; in this case, it is `+5`.
- To compute the y-intercept (`b`), plug in the value of `y` at `x = 1` and solve for `b`. In this case, `y = 2`.

```
y = m * x + b
2 = 5 * 1 + b
```

Then `b = -3`

- So the equation is

```
y = m * x + b
y = 5 * x - 3
y = 5 * count - 3
```

`count (x)`	number to print (`y`)
1	2
2	7
3	12
4	17
5	22

Another view: Slope-intercept

- Algebraically, if we always take the value of `y` at `x = 1`, then we can solve for `b` as follows:

  $$y = m * x + b$$
  $$y_1 = m * 1 + b$$
  $$y_1 = m + b$$
  $$b = y_1 - m$$

- In other words, to get the `y`-intercept, just subtract the slope from the first `y` value (`b = 2 - 5 = -3`)
  - This gets us the equation

    $$y = m * x + b$$
    $$y = 5 * x - 3$$
    $$y = 5 * count - 3$$

    (which is exactly the equation from the previous slides)

Nested `for` loop exercise

- Make a table to represent any patterns on each line.

```
....1
...2
..3
.4
5
```

line	# of dots	-1 * line	-1 * line + 5
1	4	-1	4
2	3	-2	3
3	2	-3	2
4	1	-4	1
5	0	-5	0

- To print a character multiple times, use a `for` loop.

```
for j in range(1, 5):
    print(".")          # 4 dots
```

Nested `for` loop solution

- Answer:

```
for line in range(1, 6):
    for j in range(1, (-1 * line + 5 + 1)):
        print(".", end='')
    print(line)
```

- Output:

```
....1
...2
..3
.4
5
```

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for line in range(1, 6):
    for j in range(1, -1 * line + 6):
        print(".", end='')
    for k in range(1, line):
        print(line, end='')
    print()
```

- Answer:

```
....1
...22
..333
.4444
55555
```

Nested `for` loop exercise

- Modify the previous code to produce this output:

```
....1
...2.
..3..
.4...
5....
```

- Answer:

```
for line in range(1,6):
    for j in range(1, -1 * line + 6):
        print(".", end='')
    print(line, end='')
    for j in range(1,line):
        print(".", end='')
    print()
```

Drawing complex figures

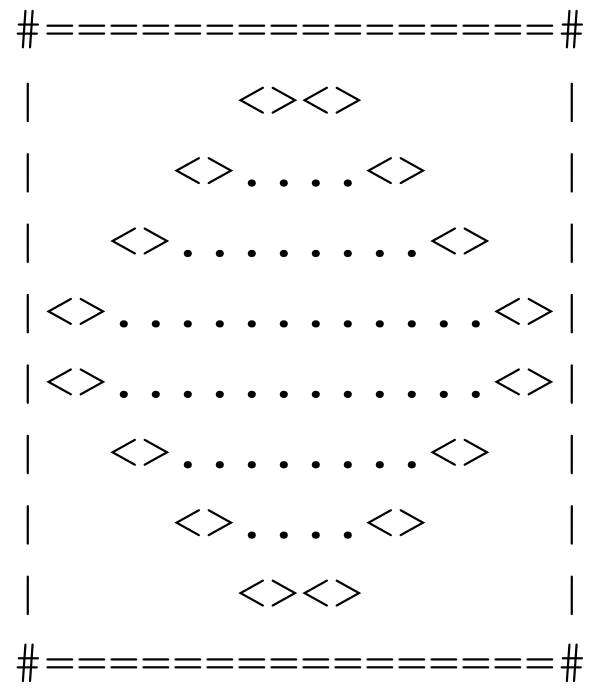
- Use nested `for` loops to produce the following output.

- Why draw ASCII art?
  - Real graphics are quite intricate
  - ASCII art has complex patterns
  - Can focus on the algorithms

```
#================#
|       <><>     |
|     <>....<>   |
|   <>........<> |
|<>............<>|
|<>............<>|
|   <>........<> |
|     <>....<>   |
|       <><>     |
#================#
```

Development strategy

- Recommendations for managing complexity:
  1. Design the program (think about steps or functions needed).
     - write an English description of steps required
     - use this description to decide the functions
  2. Create a table of patterns of characters
     - use table to write your `for` loops



1. Pseudocode

- **pseudocode**: An English description of an algorithm.

- Example: Drawing a 12 wide by 7 tall box of stars

```
print 12 stars.
for (each of 5 lines) :
    print a star.
    print 10 spaces.
    print a star.
print 12 stars.
```

```
************
*          *
*          *
*          *
*          *
*          *
************
```

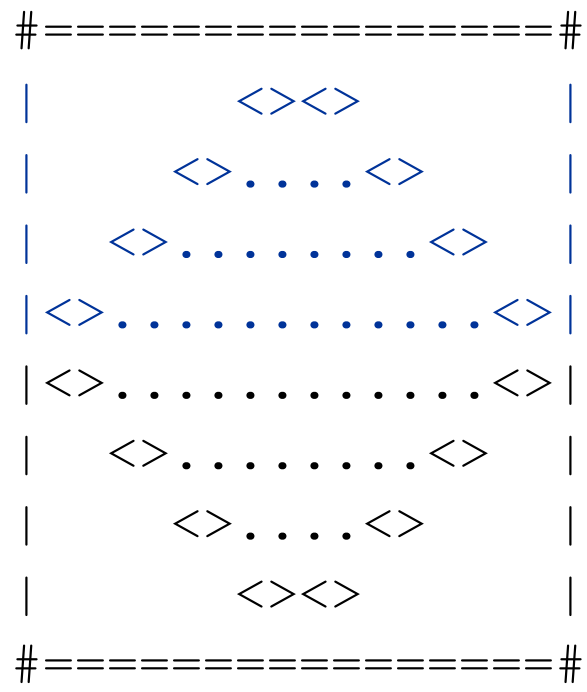
Pseudocode algorithm

1. Line
   - # , 16 =, #

2. Top half
   - |
   - spaces (decreasing)
   - <>
   - dots (increasing)
   - <>
   - spaces (same as above)
   - |

3. Bottom half (top half upside-down)

4. Line
   - # , 16 =, #



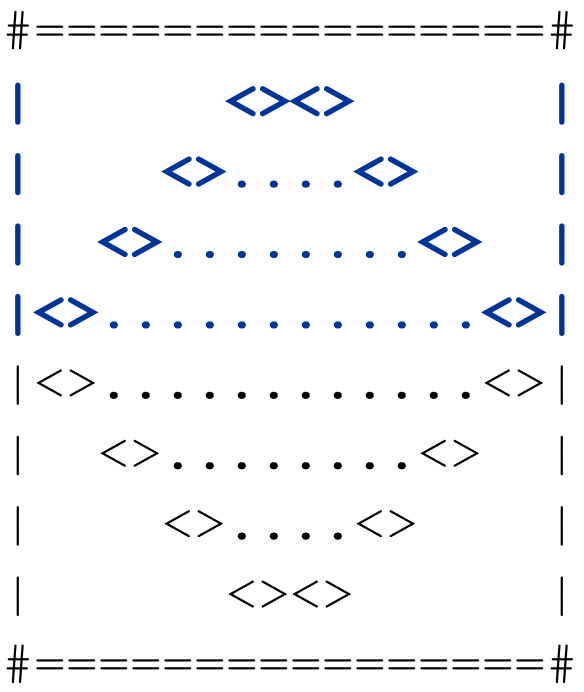
Functions from pseudocode

```
def main():
    line()
    top_half()
    bottom_half()
    line()

def top_half():
    for line in range(1, 5):
        # contents of each line

def bottom_half() {
    for line in range(1, 5):
        # contents of each line

def line():
        # ...
```

2. Tables

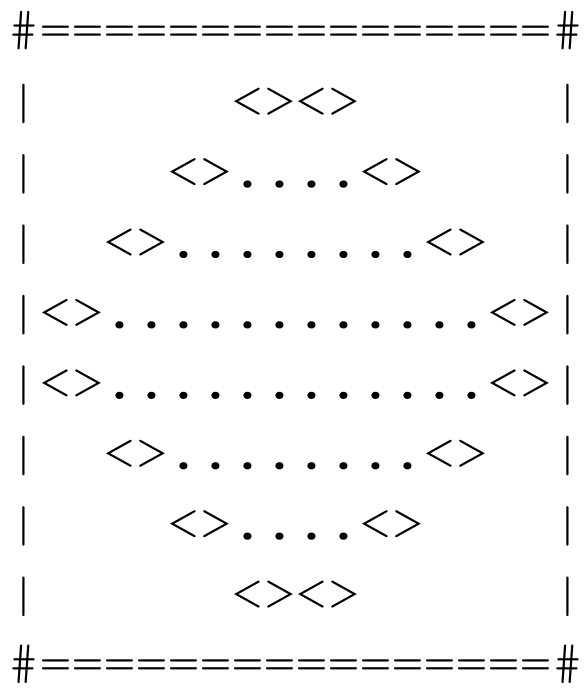
- A table for the top half:
  - Compute spaces and dots expressions from line number

line	spaces	line * -2 + 8	dots	4 * line - 4
1	6	6	0	0
2	4	4	4	4
3	2	2	8	8
4	0	0	12	12



3. Writing the code

- Useful questions about the top half:
  - Number of (nested) loops per line?



Partial solution

```
# Prints the expanding pattern of <> for the top half of the figure.
def top_half():
    for line in range(1, 5):
        print("|", end="")

        for space in range(1, line * -2 + 9):
            print(" ", end="")


        print("<>", end="")

        for dot in range(1, line * 4 - 3):
            print(".", end="")


        print("<>", end="")

        for space in range(1, line * -2 + 8):
            print(" ", end="")


        print("|")
```

Partial solution

```
# Prints the expanding pattern of <> for the top half of the figure.
def top_half():
    for line in range(1, 5):
        print("|", end="")

        for space in range(1, line * -2 + 9):
            print(" ", end="")


        print("<>", end="")

        for dot in range(1, line * 4 - 3):
            print(".", end="")


        print("<>", end="")

        for space in range(1, line * -2 + 8):
            print(" ", end="")


        print("|")
```