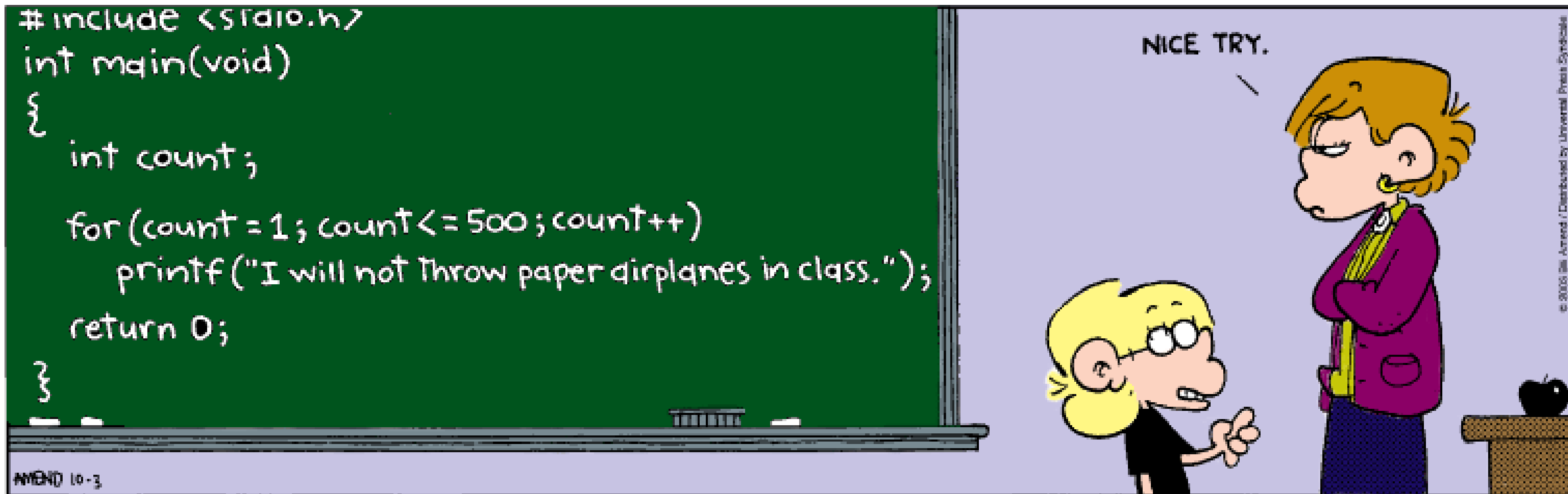
CSc 110, Autumn 2016

Lecture 5: Loop Figures and Constants

Adapted from slides by Marty Stepp and Stuart Reges

Can you write this in Python?



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
```

Drawing complex figures

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

- Why draw ASCII art?
  - Real graphics require a lot of finesse
  - ASCII art has complex patterns
  - Can focus on the algorithms

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

Development strategy

- Recommendations for managing complexity:
  1. Design the program (think about steps or methods 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. Pseudo-code

- **pseudo-code**: 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.

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

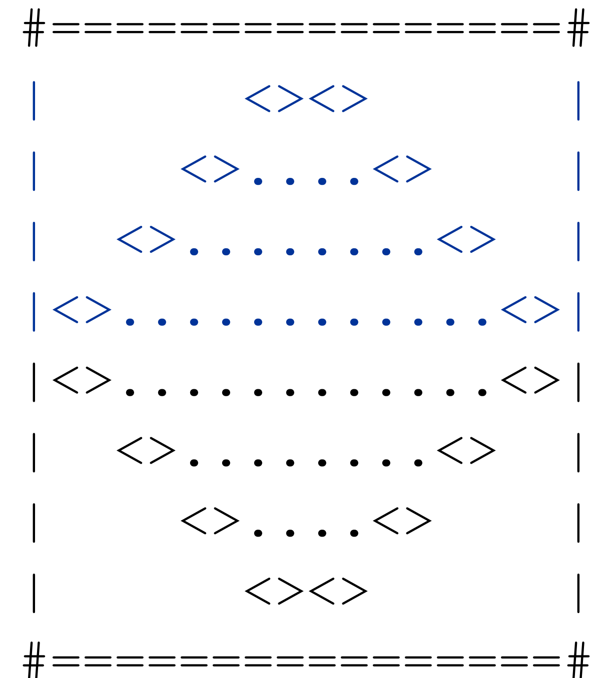
Pseudo-code algorithm

1. Line
    - # , 16 =, #

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

3. Bottom half (top half upside-down)

4. Line
    - # , 16 =, #



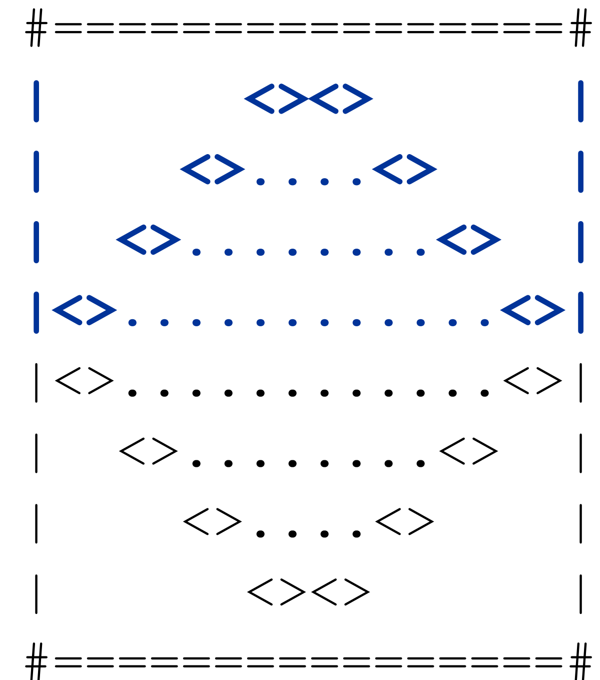
Methods from pseudocode

```
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

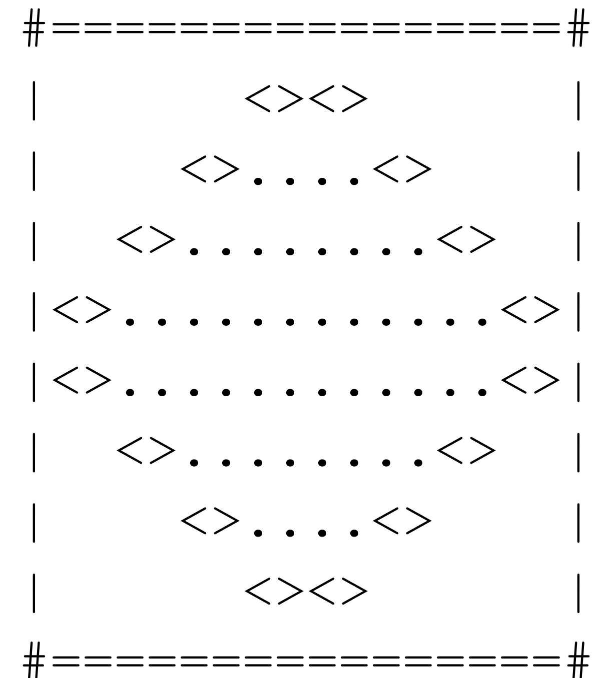
- A table for the top half:
  - Compute spaces and dots expressions from line number

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("|")
```

Class constants and scope

Scaling the mirror

- Let's modify our Mirror program so that it can scale.
  - The current mirror (left) is at size 4; the right is at size 3.

- We'd like to structure the code so we can scale the figure by changing the code in just one place.

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

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

Constants

- **constant**: A fixed value visible to the whole program.
  - value should only be set only at declaration; shouldn't be reassigned

- Syntax:
  - Just like declaring a normal variable:

    name = **value**

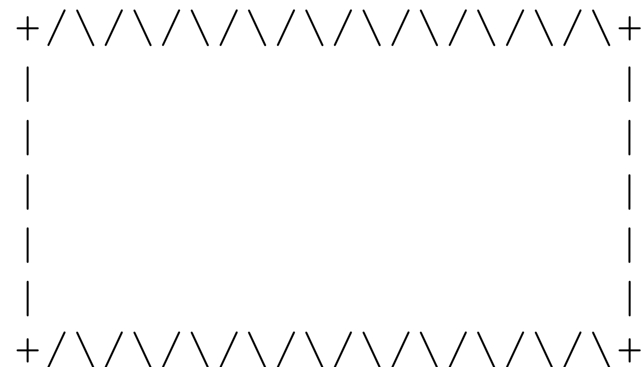
  - name is usually in ALL_UPPER_CASE

  - Examples:

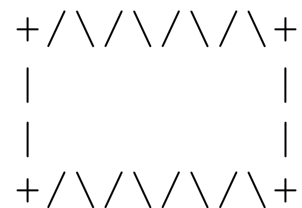
```
DAYS_IN_WEEK = 7
INTEREST_RATE = 3.5
SSN = 658234569
```

Constants and figures

- Consider the task of drawing the following scalable figure:



Multiples of 5 occur many times



The same figure at size 2

Repetitive figure code

```
def main():
    draw_line()
    draw_body()
    draw_line()

def draw_line():
    print("+", end="")
    for i in range(1, 11):
        print("/\\", end="")

    print("+")

def draw_body():
    for line in range(1, 6):
        print("|", end="")
        for spaces in range(1, 21):
            print(" ", end="")

        print("|")
```

Adding a constant

```
HEIGHT = 5
def main():
    draw_line()
    draw_body()
    draw_line()

def draw_line():
    print("+", end="")
    for i in range(1, HEIGHT * 2 + 1):
        print("/\\", end="")

    print("+")

def draw_body():
    for line in range(1, HEIGHT + 1):
        print("|", end="")
        for spaces in range(1, HEIGHT * 4 + 1):
            print(" ", end="")

        print("|")
```

Complex figure w/ constant

- Modify the Mirror code to be resizable using a constant.

A mirror of size 4:

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

A mirror of size 3:

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

Loop tables and constant

- Let's modify our loop table to use `SIZE`
  - This can change the amount added in the loop expression

SIZE	line	spaces		dots	
4	1,2,3,4	6,4,2,0		0,4,8,12	
3	1,2,3	4,2,0		0,4,8	

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

Partial solution

```
SIZE = 4;
# Prints the expanding pattern of <> for the top half of the figure.
def top_half() {
    for line in range(1, SIZE):
        print("|", end="")
        for space in range(1, line * -2 + (2*SIZE) + 1):
            print(" ", end="")

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

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

        print("|")
```

Observations about constant

- The constant can change the "intercept" in an expression.
  - Usually the "slope" is unchanged.

```
SIZE = 4;

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

- It doesn't replace *every* occurrence of the original value.

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