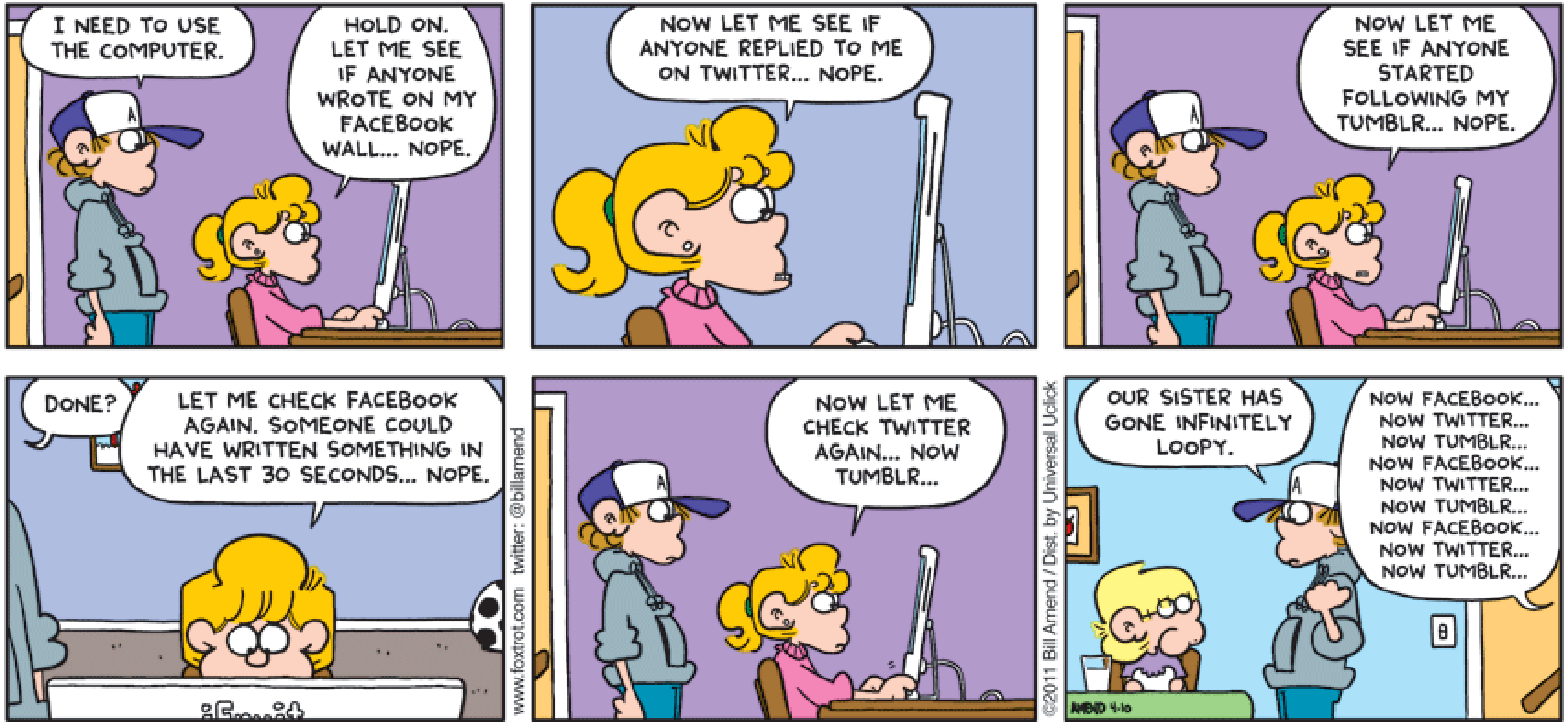
CSc 110, Autumn 2016

Lecture 4: The `for` Loop

Adapted from slides by Marty Stepp and Stuart Reges


I NEED TO USE THE COMPUTER.
HOLD ON. LET ME SEE IF ANYONE WROTE ON MY FACEBOOK WALL... NOPE.
NOW LET ME SEE IF ANYONE REPLIED TO ME ON TWITTER... NOPE.
NOW LET ME SEE IF ANYONE STARTED FOLLOWING MY TUMBLR... NOPE.
DONE?
LET ME CHECK FACEBOOK AGAIN. SOMEONE COULD HAVE WRITTEN SOMETHING IN THE LAST 30 SECONDS... NOPE.
www.foxtrot.com twitter: @billamend
NOW LET ME CHECK TWITTER AGAIN... NOW TUMBLR...
©2011 Bill Amend / Dist. by Universal Uclick
OUR SISTER HAS GONE INFINITELY LOOPY.
NOW FACEBOOK... NOW TWITTER... NOW TUMBLR... NOW FACEBOOK... NOW TWITTER... NOW TUMBLR... NOW FACEBOOK... NOW TWITTER... NOW TUMBLR...
AMEND 4-10

Repetition with `for` loops

- So far, repeating an action results in redundant code:

```
makeBatter()
bakeCookies()
bakeCookies()
bakeCookies()
bakeCookies()
bakeCookies()
frostCookies()
```

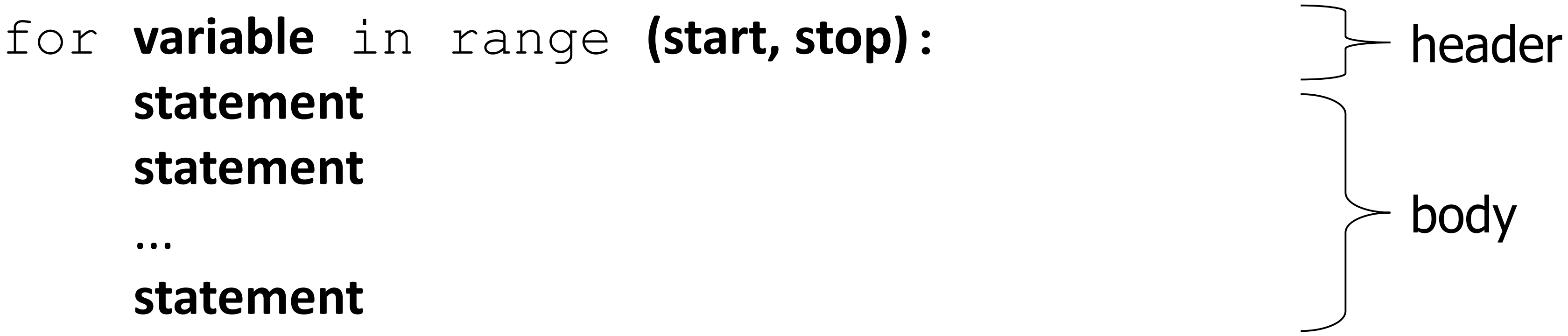
- Python's **`for` loop** statement performs a task many times.

```
mixBatter()

for i in range(1, 6):     # repeat 5 times
    bakeCookies()

frostCookies()
```

for loop syntax



```
for variable in range (start, stop):
        statement
        statement
        ...
        statement
```

- Set the variable equal to the start value
- Repeat the following:
  - Check if the **variable** is less than the stop. If not, stop.
  - Execute the **statement**s.
  - Increase the variable's value by 1.

Control structures

- **Control structure**: a programming construct that affects the flow of a program's execution

- Controlled code may include one or more statements

- The for loop is an example of a looping control structure

Repetition over a range

```
print("1 squared = " + str(1 * 1))
print("2 squared = " + str(2 * 2))
print("3 squared = " + str(3 * 3))
print("4 squared = " + str(4 * 4))
print("5 squared = " + str(5 * 5))
print("6 squared = " + str(6 * 6))
```

- Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for i in range(1, 7):
    print(str(i) + " squared = " + str(i * i));
```

- "For each integer **i** from 1 through 6, print ..."

Loop walkthrough

```
for i in range(1, 5):
    print(str(i) + " squared = " + str(i * i))

print("Whoo!")
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```

Multi-line loop body

```
print("+-----+")
for i in range(1, 4):
    print("\\     /")
    print("/     \\")
print("+-----+")
```

- Output:

```
+-----+
\     /
/     \
\     /
/     \
\     /
/     \
+-----+
```

Expressions for counter

```
highTemp = 5
for i in range(-3, high_temp // 2 + 1):
    print(i * 1.8 + 32)
```

- Output:

```
26.6
28.4
30.2
32.0
33.8
35.6
```

Rocket Exercise

- Write a method that produces the following output:

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

print('', end='')

- Adding `, end=''` allows you to print without moving to the next line
  - allows you to print partial messages on the same line

```
highTemp = 5
for i in range(-3, int(highTemp / 2 + 1)):
    print(i * 1.8 + 32, end=' ')
```

- Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

  - Either concatenate `'   '` to separate the numbers or set `end='   '`

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

```
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?

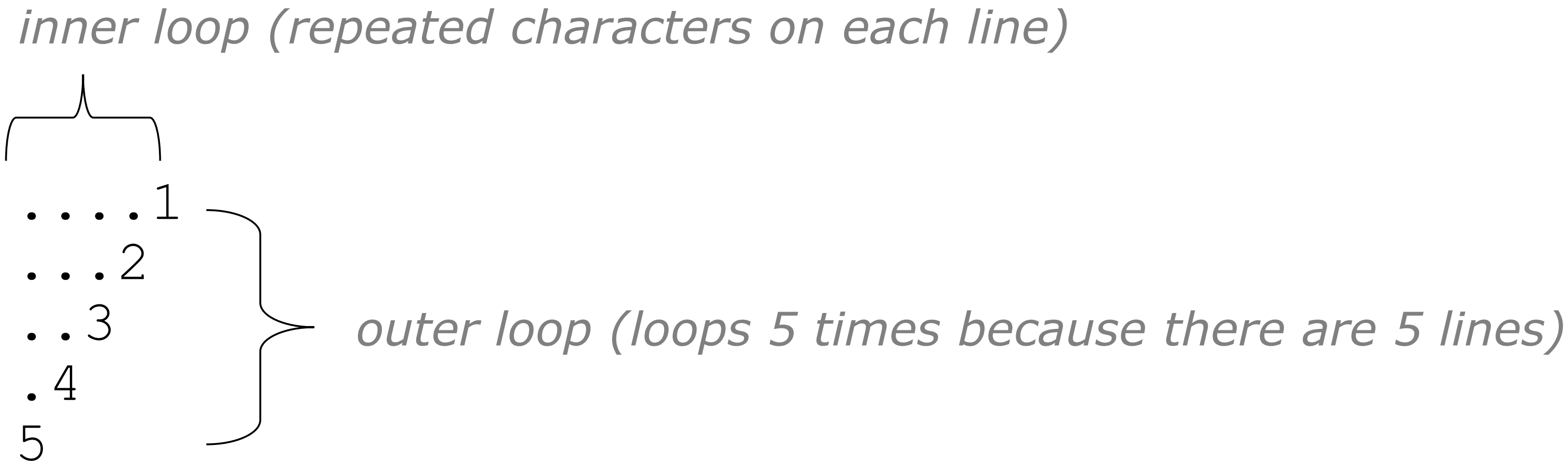
```
for i in range(1, 6):
    for j in range(1, i + 1):
        print(i, end="")
    print()
```

- Output:

```
1
22
333
4444
55555
```

Complex lines

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



inner loop (repeated characters on each line)

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

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:
    - some dots (0 dots on the last line), 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:

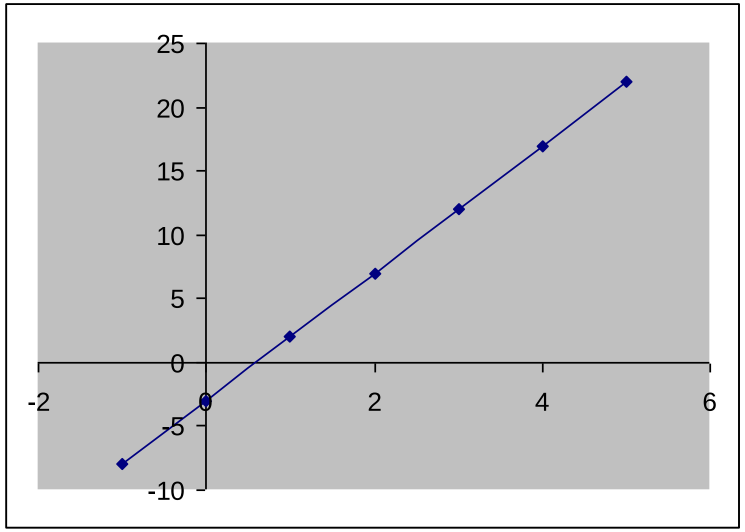
```
17 13 9 5 1
```

- 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()
```

Modify-and-assign operators

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += **value**;	**variable** = **variable** + **value**;
variable -= **value**;	**variable** = **variable** - **value**;
variable *= **value**;	**variable** = **variable** * **value**;
variable /= **value**;	**variable** = **variable** / **value**;
variable %= **value**;	**variable** = **variable** % **value**;

```
x += 3;                 # x = x + 3;

gpa -= 0.5;             # gpa = gpa - 0.5;

number *= 2;            # number = number * 2;
```