SAMS
Programming - Section C

Week 2 - Lecture 2:
More strings + Nested loops + Style

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

July 12, 2017

On the menu today

Wrap up strings

Nested loops

Style

Wrap up strings

String literals

x = “#FeelTheBern” → string literal

x = ‘#FeelTheBern’ single-quotes

x = ‘‘‘#FeelTheBern’’’ triple single-quotes

x = “““#FeelTheBern””” triple double-quotes

What are the differences between these?

String literals

Single-quotes and **double-quotes** work similarly.

print(“hello world”)	hello world
print(‘hello world’)	hello world
print(“Bernie said: “hello world”.”)	Syntax error
print(‘Bernie said: “hello world”.’)	Bernie said: “hello world”.
print(“Bernie said: ‘hello world’.”)	Bernie said: ‘hello world’.
print(“Hello World”)	Syntax error

String literals

Use **triple quotes** for multi-line strings.

```
print(“““hello
world”””)
```

```
hello
world
```

```
x = ‘‘‘#FeelTheBern
!’’’
```

```
print(x)
```

```
#FeelTheBern
!
```

newline character

What value does x really store?

‘#FeelTheBern**\n**!’

String literals

\n newline **\t tab**

```
x = “#FeelTheBern\n!”

print(x)
```

```
#FeelTheBern
!
```

```
x = “#FeelTheBern\t!”

print(x)
```

```
#FeelTheBern      !
```

String literals

Escape characters: use \

```
print(“The newline character is \n.”)
```

```
The newline character is
.
```

```
print(“The newline character is \\n.”)
```

```
The newline character is \n.
```

```
print(“He said: \“hello world\”.”)
```

```
He said:“hello world”.
```

String literals

Second functionality of \ : ignore newline

```
print(“‘#FeelTheBern
!”’)
```

```
#FeelTheBern
!
```

```
print(“‘#FeelTheBern \
!”’)
```

```
#FeelTheBern !
```

```
print(‘#FeelTheBern \
!’)
```

```
#FeelTheBern !
```

Built-in constants

```
import string

print(string.ascii_letters)

print(string.ascii_lowercase)

print(string.ascii_uppercase)

print(string.digits)

print(string.punctuation)

print(string.printable)

print(string.whitespace)

print(“\n” in string.whitespace)
```

Example

```
import string


def isLowercase(char):
    return (char in string.ascii_lowercase)


def isWhitespace(char):
    return (char in string.whitespace)
```

Built-in string methods

Method: a function applied “directly” on an object/data

Example: there is a string method called upper( ),
it works like toUpper( ) from the HW.

```
s = “hey you!”

print(upper(s))        ERROR:  not used like a function.

print(s.upper())       HEY YOU!
```

```
s.upper()       is basically like
upper(s)          (if upper was a function)
```

Built-in string methods

Method: a function applied “directly” on an object/data

Example: there is a string method called count( ):

```
s = “hey hey you!”

print(s.count(“hey”))      2
```

`s.count(“hey”)` is basically like
`count(s, “hey”)` (if count was a function)

Built-in string methods

isupper

islower

isdigit

isalnum

isalpha

isspace

upper

lower

replace

strip

count

startswith

endswith

find

Built-in string methods

split and splitlines

```
names = “Alice,Bob,Charlie,David”

for name in names.split(“,”):
    print(name)
```

```
Alice
Bob
Charlie
David
```

returns [“Alice”, “Bob”, “Charlie”, “David”]

Built-in string methods

split and splitlines

```
s.splitlines()  ≈  s.split(“\n”)
```

```
quotes = “““\
Dijkstra: Simplicity is prerequisite for reliability.
Knuth: If you optimize everything, you will always be unhappy.
Dijkstra: Perfecting oneself is as much unlearning as it is learning.
Knuth: Beware of bugs in the above code; I have only proved it correct, not tried it.
Dijkstra: Computer science is no more about computers than astronomy is about telescopes.
”””
```

```
for line in quotes.splitlines():
    if (line.startswith(“Knuth”)):
        print(line)
```

String formatting

```
team = “Steelers”
numSB = 6
s =  “The ” + team + “ have won ” +  numSB + “ Super Bowls.”
```

String formatting

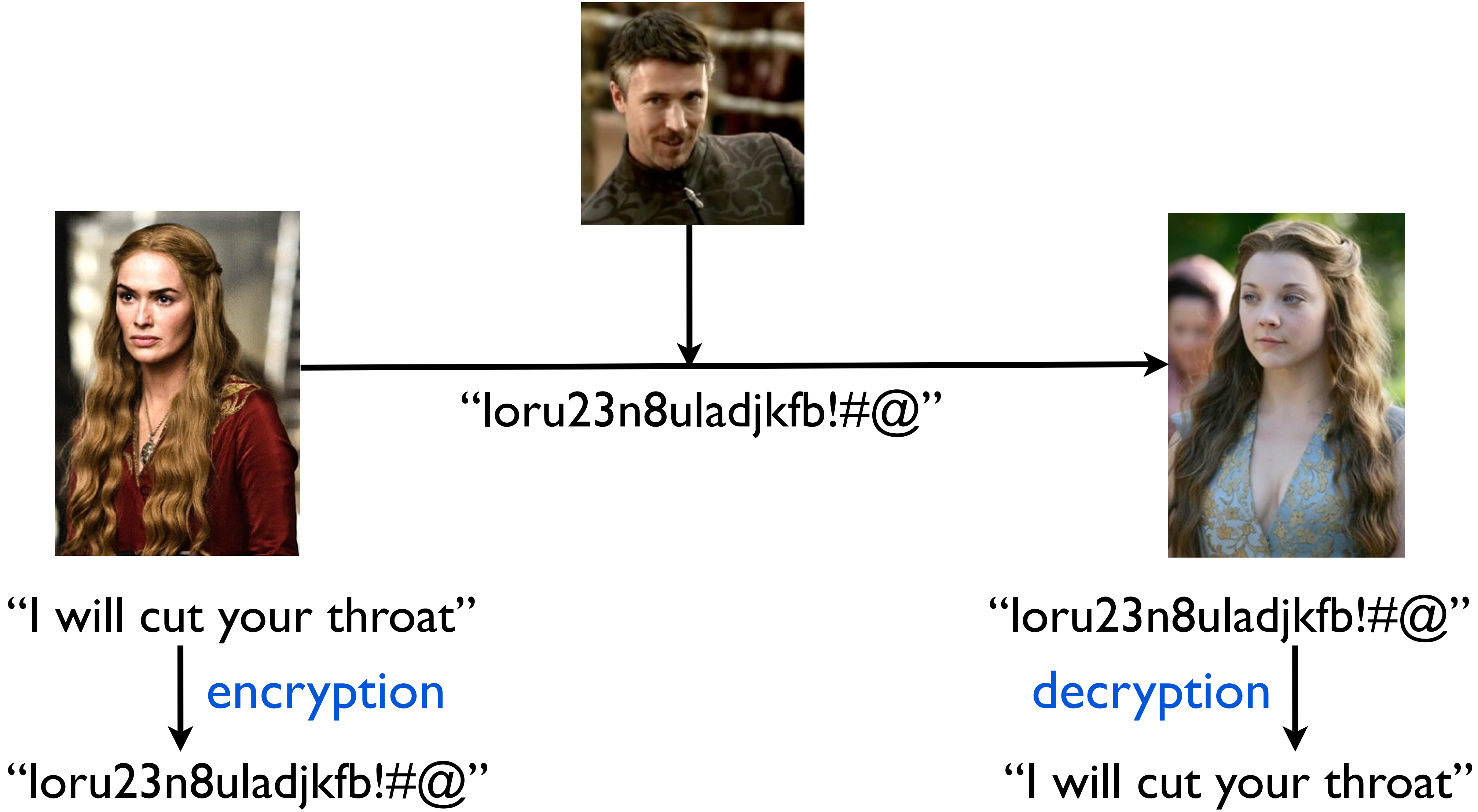
```
team = “Steelers”
numSB = 6
s =  “The ” + team + “ have won ” + str(numSB) + “ Super Bowls.”
```

```
team = “Steelers”
numSB = 6
s =  “The %s have won %d Super Bowls” % (team, numSB)
```

string   decimal

```
print(s)      The Steelers have won 6 Super Bowls
```

Example: Cryptography



"loru23n8uladjkfb!#@"

"I will cut your throat"

encryption

"loru23n8uladjkfb!#@"

"loru23n8uladjkfb!#@"

decryption

"I will cut your throat"

Example: Caesar shift

Encrypt messages by shifting each letter a certain number of places.

<u>Example</u>: shift by 3

a b c d e f g h i j k l m n o p q r s t u v w x y z
↓ ↓
d e f g h i j k l m n o p q r s t u v w x y z a b c

(similarly for capital letters)

“Dear Math, please grow up and solve your own problems.”

↓

“Ghdu Pdwk, sohdvh jurz xs dqg vroyh brxu rzq sureohpv.”

Write functions to encrypt and decrypt messages.

Example: Caesar shift

```
def encrypt(message, shiftNum):
    result = “”
    for char in message:
        result += shift(char, shiftNum)
    return result


def shift(c, shiftNum):
    shiftNum %= 26
    if (not c.isalpha()):
        return c
    alph = string.ascii_lower if (c.islower()) else string.ascii_upper
    shiftedAlph = alph[shiftNum:] + alph[:shiftNum]
    return shiftedAlph[alph.find(c)]
```

Example: Caesar shift

```
def shift2(c, shiftNum):
    shiftNum %= 26
    if (‘A’ <= c <= ‘Z’):
        if (ord(c) + shiftNum > ord(‘Z’)):
            return chr(ord(c) + shiftNum - 26)
        else:
            return chr(ord(c) + shiftNum)
    elif (‘a’ <= c <= ‘z’):
        if (ord(c) + shiftNum > ord(‘z’)):
            return chr(ord(c) + shiftNum - 26)
        else:
            return chr(ord(c) + shiftNum)
    else:
        return c
```

Code repetition

Exercise: Rewrite avoiding the repetition

Tangent: Private-Key Cryptography

Cryptography before WWII

Tangent: Private-Key Cryptography

Cryptography before WWII

“#dfg%y@d2hSh2$&”

“I will cut your throat”

“#dfg%y@d2hSh2$&”

“#dfg%y@d2hSh2$&”

“I will cut your throat”

Tangent: Private-Key Cryptography

Cryptography before WWII

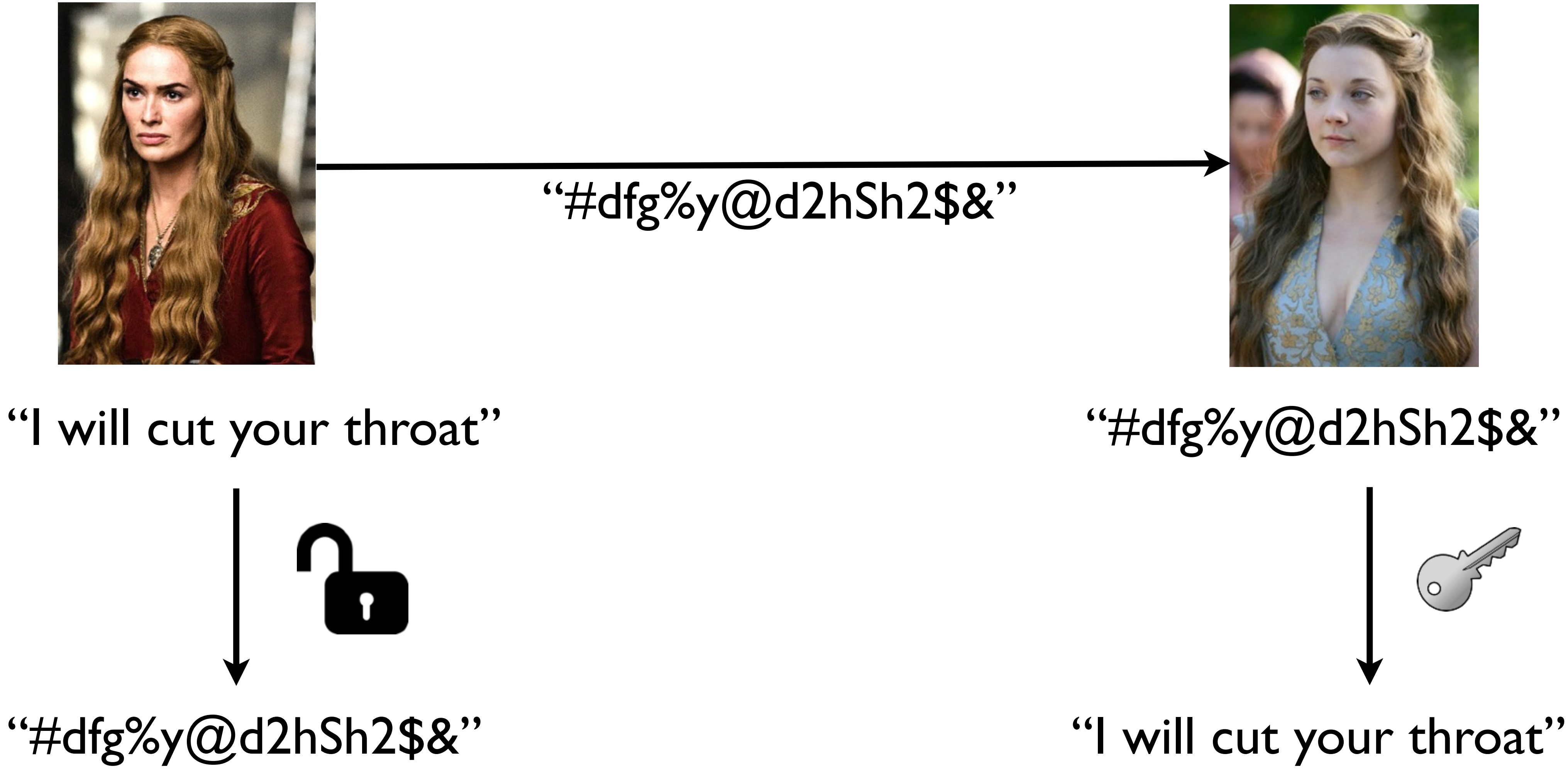
there must be a secure way of exchanging the key

Tangent: Public-Key Cryptography

Cryptography after WWII

Tangent: Public-Key Cryptography

Cryptography <u>after</u> WWII



"#dfg%y@d2hSh2$&"

"I will cut your throat"

"#dfg%y@d2hSh2$&"

"#dfg%y@d2hSh2$&"

"I will cut your throat"

Tangent: The factoring problem

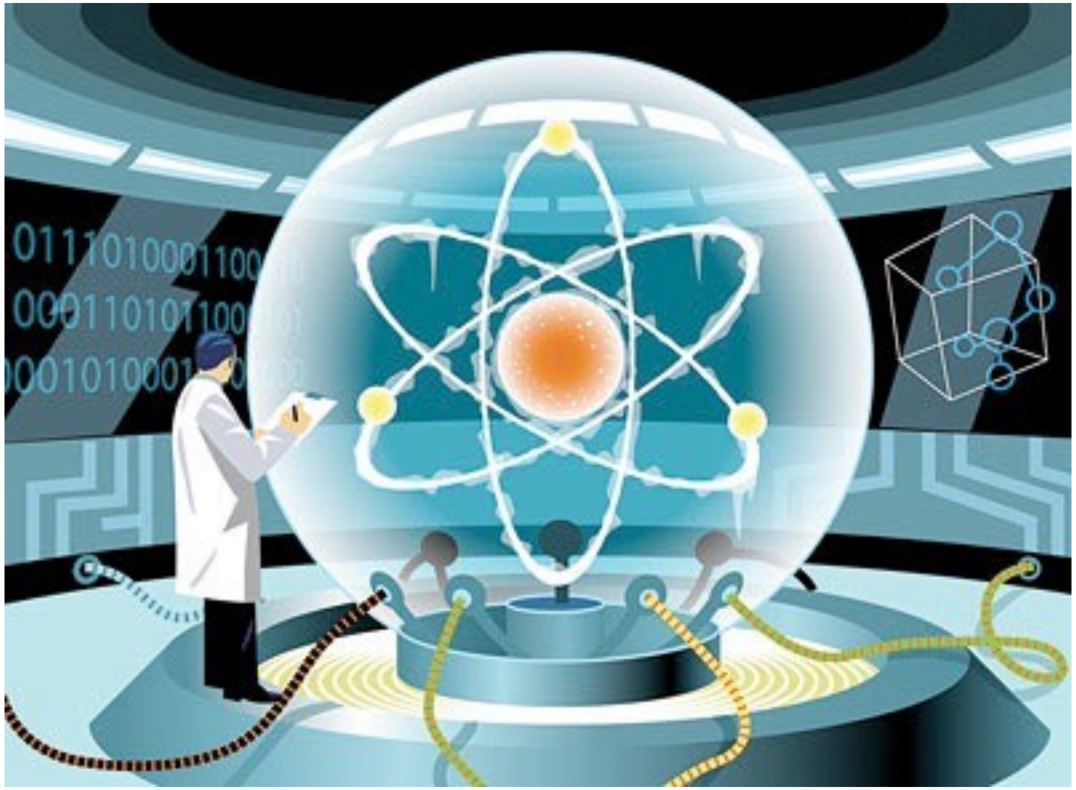
If there is an efficient program to solve
the factoring problem

↓

can ***break*** public-key crypto systems
used over the internet

Fun fact: *Quantum computers* can factor large numbers efficiently!

Tangent: What is a quantum computer?



Information processing using quantum physics.

Nested loops

My first ever program

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

Nested loops

Many situations require one loop inside another loop.

```
for y in range(10):
    for x in range(8):
        # Body of the nested loop
```

Nested loops

Many situations require one loop inside another loop.

```
for y in range(10):
    for x in range(8):
        print(“Hello”)
```

How many times will “Hello” get printed?

Nested loops

Many situations require one loop inside another loop.

```
for y in range(4):
    for x in range(y):
        print(“Hello”)
```

y	# iterations of inner loop
0	0
1	1
2	2
3	3

How many times will “Hello” get printed?

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

height = 4, width = 3

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

```
Repeat 4 times:
    - Print a row (3 stars)
```

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

height = 4, width = 3

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

```
Repeat 4 times:
    Repeat 3 times:
        - Print a single star
    Skip a line
```

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

height = 4, width = 3

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

```
for row in range(4):
    for col in range(3):
        print(“*”, end=“ ”)
    print()
```

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

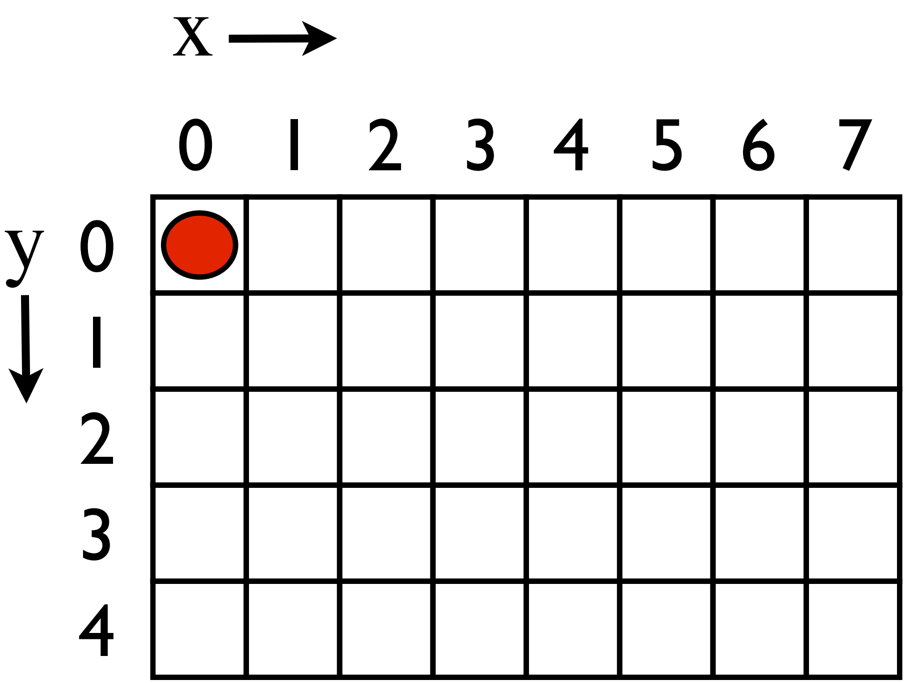
height = 4, width = 3

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

```
def printRectangle(height, width):
    for row in range(height):
        for col in range(width):
            print(“*”, end= “ ”)
        print()
```

Nested loops

```
for y in range(5):
    for x in range(8):
        # Body of the nested loop
```



Example

```
for y in range(4):
    for x in range(5):
        print(“( %d , %d )” % (x, y)), end=“ ”)
    print()
```

```
             x →

y       ( 0 ,0 ) ( 1 ,0 ) ( 2 ,0 ) ( 3 ,0 ) ( 4 ,0 )
↓       ( 0 , 1 ) ( 1 , 1 ) ( 2 , 1 ) ( 3 , 1 ) ( 4 , 1 )
        ( 0 , 2 ) ( 1 , 2 ) ( 2 , 2 ) ( 3 , 2 ) ( 4 , 2 )
        ( 0 , 3 ) ( 1 , 3 ) ( 2 , 3 ) ( 3 , 3 ) ( 4 , 3 )
```

Example

```
for y in range(4):
    for x in range(y):
        print(“( %d , %d )” % (x, y)), end=“ ”)
    print()
```

```
\n
( 0 , 1 )
( 0 , 2 ) ( 1 , 2 )
( 0 , 3 ) ( 1 , 3 ) ( 2 , 3 )
```

Example

```
for y in range(1, 10):
    for x in range(1, 10):
        print(y*x, end=“ ”)
    print()
```

Multiplication table

```
for y in range(1, 10):
    for x in range(1, 10):
        print(y*x, end=“ ”)
    print()
```

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

A trick to get rid of nested loops

Write a function for the inner loop.

Example: Write a function that:

- Gets an integer height as input
- Prints a right-angled triangle of that height

height = 5

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

```
def printStars(n):
    for x in range(n):
        print(“*”, end=“”)

def printTriangle(height):
    for x in range(height):
        printStars(   ?   )
    print()
```

A trick to get rid of nested loops

Write a function for the inner loop.

Example: Write a function that:

- Gets an integer height as input
- Prints a right-angled triangle of that height

```
height = 5
```

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

```
def printStars(n):
    for x in range(n):
        print(“*”, end=“”)

def printTriangle(height):
    for x in range(height):
        printStars(height - x)
    print()
```

A common nested loop

Input: a string s

Output: True if s contains a character more than once.
False otherwise.

```
def hasDuplicates(s):
    for i in range(len(s)-1):
        for j in range(i+1, len(s)):
            if(s[i] == s[j]): return True
    return False
```

Style

From lecture 1

What you will learn in this course:

1. How to think like a computer scientist.

2. Principals of good programming.

3. Programming language: Python

From lecture 1

2. Principals of good programming.

Is your code easy to read? easy to understand?

Can it be reused easily? extended easily?

Is it easy to fix errors (bugs)?

Are there redundancies in the code?

Style: Summary

better style = better code

= a better world

Strong correlation between bad style and # bugs

Good style **--->** saves money

Good style **--->** saves lives

Style guides

- Official Python Style Guide

- Google Python Style Guide

- Our own Style Guide

Our Style Guidelines

Comments

Concise, clear, informative comments when needed.

Our Style Guidelines

Comments

Ownership Good

```
# Name: Anil Ada
# Andrew id: aada
# Section: C
```

Our Style Guidelines

Comments

Before function bodies (if not obvious)   Good

```
def foo():
    “‘This function returns the answer to the ultimate question
    of life, the universe, and everything.’”
    return 42
```

Our Style Guidelines

Comments

Before a logically connected block of code

Good

```
def foo():
    …
    …
    # Compute the distance between Earth and its moon.
    …
    …
```

Our Style Guidelines

Comments

Bad

```
x = 1     # Assign 1 to x
```

Our Style Guidelines

Comments

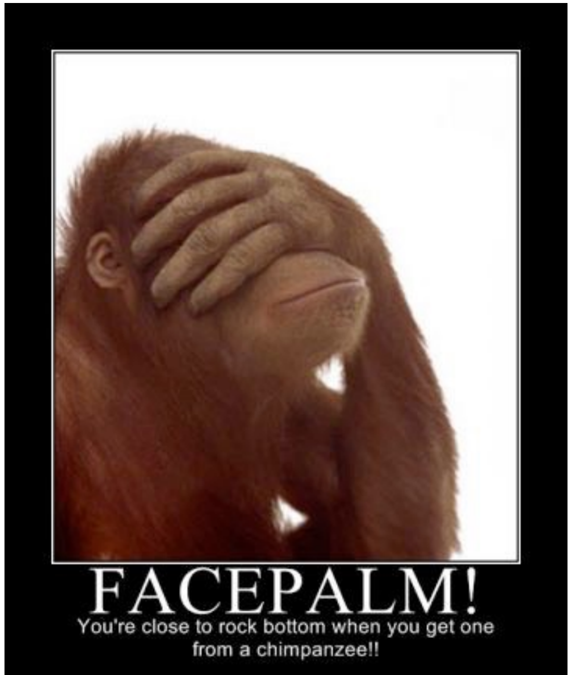
Very Bad

```
x = 1    # Assign 10 to x
```

Our Style Guidelines

Comments

'''This function takes as input a thing that represents the thing that measures how long it takes to go from the center of a round circle to the outer edge of it. I learned in elementary school that.........
The number PI does not really have anything to do with apple pie, although I kind of wish it did because it's really delicious. My grandma makes great pies.'''


FACEPALM!
You're close to rock bottom when you get one from a chimpanzee!!

Our Style Guidelines

Helper functions

Use helper functions liberally!!!

No function can contain more than 20 lines.
(25 lines for functions using graphics)

Our Style Guidelines

Test functions

Each function should have a corresponding test function!!!

exceptions: graphics, functions with no returned value

Our Style Guidelines

Clarity

```
def abs(n):
    return (n < 0)*(-n) + (n >= 0)*(n)
```

Bad style!

```
def abs(n):
    if (n < 0):
        return -n
    else:
        return n
```

Our Style Guidelines

Meaningful variable/function names

No more a, b, c, d, u, ww, pt, qr, abc

Use mixedCase.

Bad variable names

```
a                thething
anonymous        anilsucks
```

Good variable names

```
length       degreesInFahrenheit
counter      theMessageToTellAnilHeSucks
```

Our Style Guidelines

“Numbered” variables

count0
count1
count2
count3
count4
count5
count6
count7
count8
count9

Use lists and/or loops

Our Style Guidelines

Magic numbers

Hides logic. Harder to debug.

```
def toUpperCaseLetter(c):
    if (“a” <= c <= “z”):
        return chr(ord(c) - 32)
    return c
```

magic number

Our Style Guidelines

Magic numbers

Hides logic. Harder to debug.

```
def shift(c, shiftNum):
    shiftNum %= 26
    if (not c.isalpha()):
        return c
    alph = string.ascii_lower if (c.islower())  else string.ascii_upper
    shifted_alph = alph[shiftNum:] + alph[:shiftNum]
    return shifted_alph[alph.find(c)]
```

→ magic number

Our Style Guidelines

Magic numbers

Hides logic. Harder to debug.

```
def shift(c, shiftNum):
→   alphabetSize = 26
    shiftNum %= alphabetSize
    if (not c.isalpha()):
        return c
    alph = string.ascii_lower if (c.islower()) else string.ascii_upper
    shifted_alph = alph[shiftNum:] + alph[:shiftNum]
    return shifted_alph[alph.find(c)]
```

Our Style Guidelines

Formatting

- max 80 characters per line
- proper indentation (use 4 spaces, not tab)
- one or two blank lines between functions
- one blank line to separate logical sections

Our Style Guidelines

Others

Efficiency

Global variables

Duplicate code

Dead code

Meaningful User Interface (UI)

Other guidelines as described in course notes