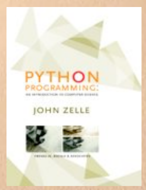
Python Programming: An Introduction to Computer Science

Chapter 1

Computers and Programs

Modified by Dan Fleck



PYTHON PROGRAMMING:

JOHN ZELLE

Objectives

- Introduction to the class
- Why we program and what that means
- Introduction to the Python programming language

What we'll learn in this class

- How to solve problems using computers
- How computer programs work
- How to write computer programs in Python
- How to debug (fix problems with) computer programs
- How to test your programs

Lab 01

- Logon to Blackboard: courses.gmu.edu
- Go to CS112
- Click on assignments
- View the assignment and complete it
- Also, activate your GMU Email! You can easily forward it to Yahoo or Gmail. Make sure you check the account daily! https://mail.gmu.edu

The Universal Machine

- A modern computer can be defined as “a machine that stores and manipulates information under the control of a changeable program.”
- Two key elements:
  - Computers are devices for manipulating information.
  - Computers operate under the control of a changeable program.

What is a computer program

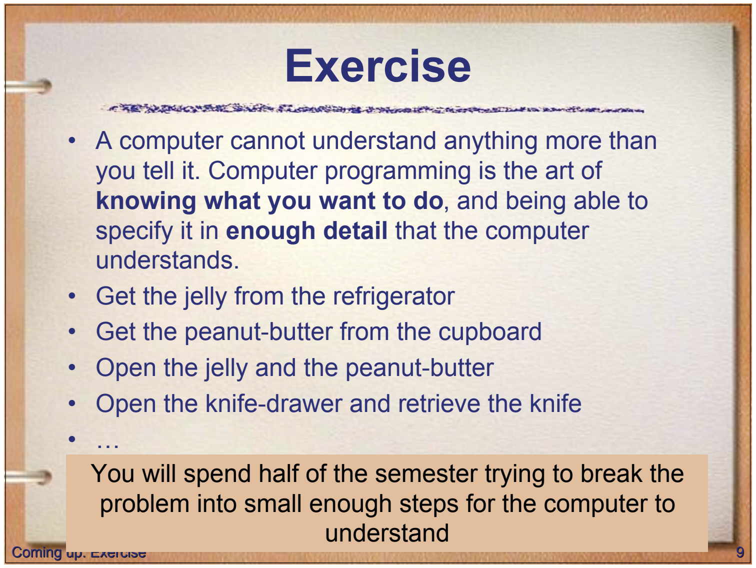
- A detailed, step-by-step set of instructions telling a computer what to do.
- If we change the program, the computer performs a different set of actions or a different task.
- The machine stays the same, but the program changes!
- The program is *executed* or carried out by the machine

Exercise

- Write the step-by-step instructions for creating a peanut butter and jelly sandwich

Exercise

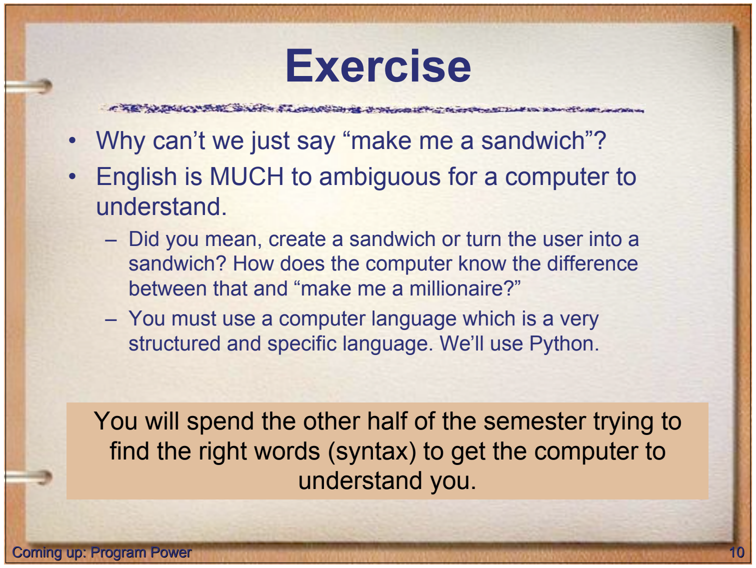
- Write the step-by-step instructions for creating a peanut butter and jelly sandwich
- Did you write:
  - Spread peanut-butter on bread
  - Spread jelly on bread
  - Put pieces of bread together



Exercise

- A computer cannot understand anything more than you tell it. Computer programming is the art of **knowing what you want to do**, and being able to specify it in **enough detail** that the computer understands.
- Get the jelly from the refrigerator
- Get the peanut-butter from the cupboard
- Open the jelly and the peanut-butter
- Open the knife-drawer and retrieve the knife
- …

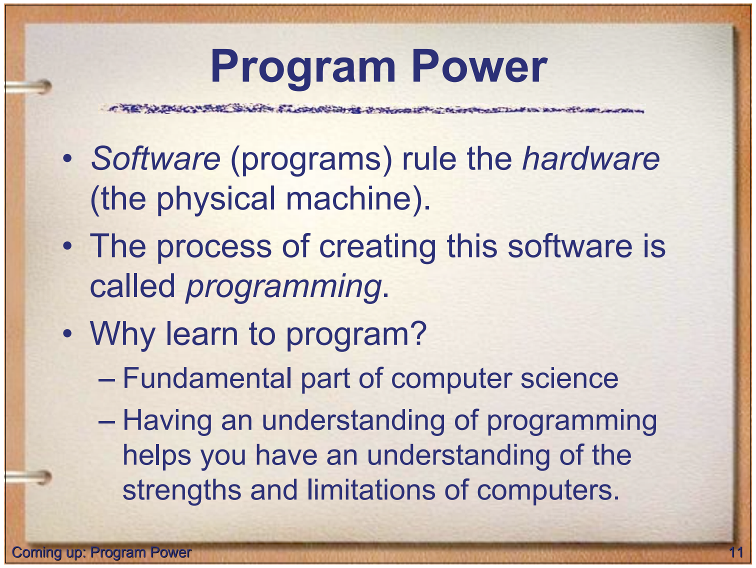
You will spend half of the semester trying to break the problem into small enough steps for the computer to understand



Exercise

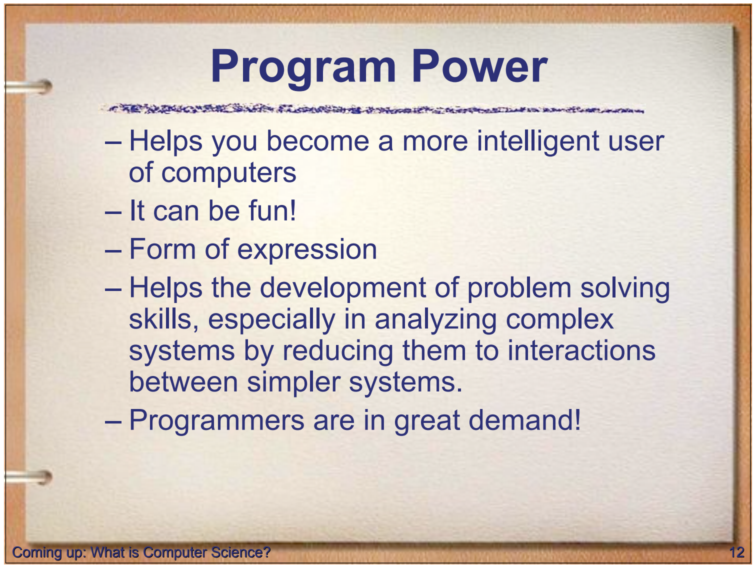
- Why can't we just say "make me a sandwich"?
- English is MUCH to ambiguous for a computer to understand.
  - Did you mean, create a sandwich or turn the user into a sandwich? How does the computer know the difference between that and "make me a millionaire?"
  - You must use a computer language which is a very structured and specific language. We'll use Python.

You will spend the other half of the semester trying to find the right words (syntax) to get the computer to understand you.



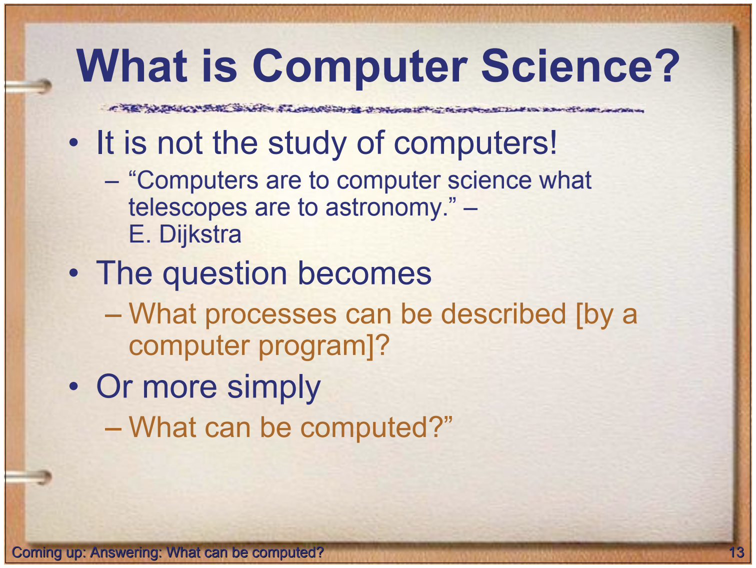
Program Power

- *Software* (programs) rule the *hardware* (the physical machine).
- The process of creating this software is called *programming*.
- Why learn to program?
  - Fundamental part of computer science
  - Having an understanding of programming helps you have an understanding of the strengths and limitations of computers.



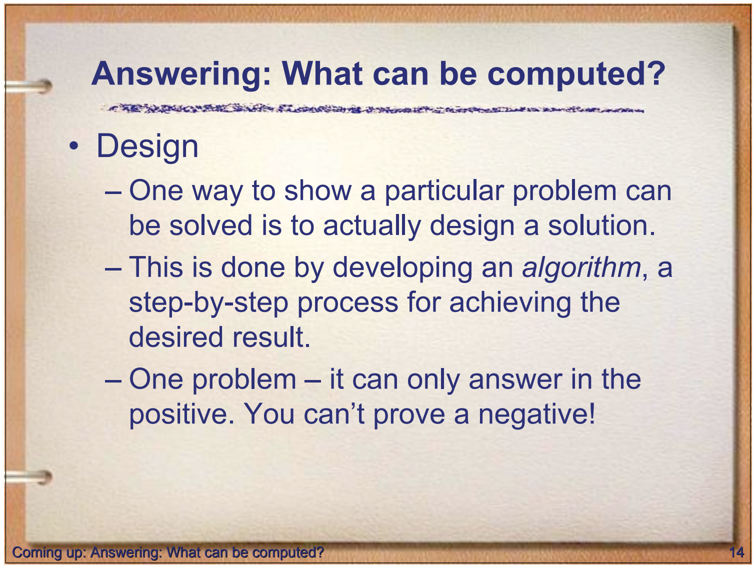
Program Power

- Helps you become a more intelligent user of computers
- It can be fun!
- Form of expression
- Helps the development of problem solving skills, especially in analyzing complex systems by reducing them to interactions between simpler systems.
- Programmers are in great demand!



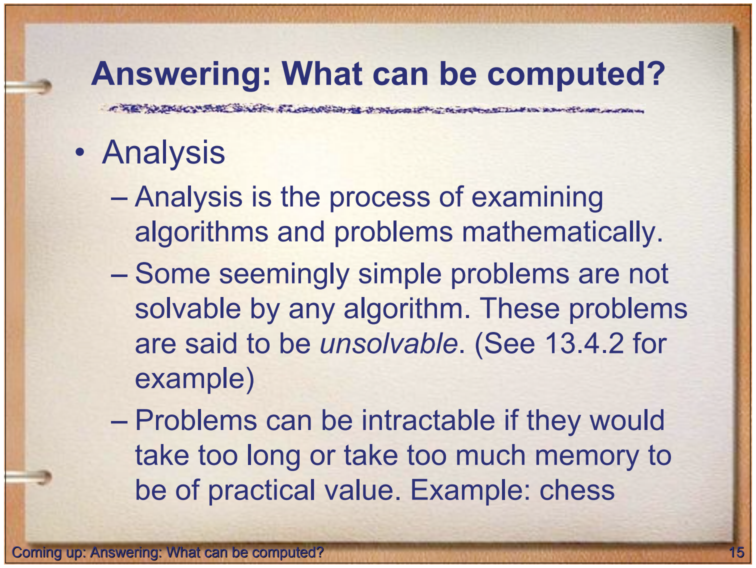
What is Computer Science?

- It is not the study of computers!
  - “Computers are to computer science what telescopes are to astronomy.” – E. Dijkstra
- The question becomes
  - What processes can be described [by a computer program]?
- Or more simply
  - What can be computed?”



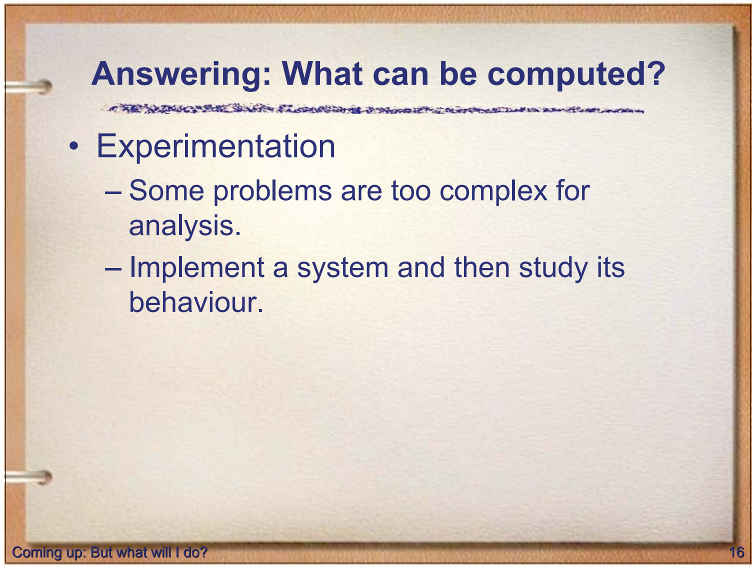
Answering: What can be computed?

- Design
  - One way to show a particular problem can be solved is to actually design a solution.
  - This is done by developing an *algorithm*, a step-by-step process for achieving the desired result.
  - One problem – it can only answer in the positive. You can’t prove a negative!



Answering: What can be computed?

- Analysis
  - Analysis is the process of examining algorithms and problems mathematically.
  - Some seemingly simple problems are not solvable by any algorithm. These problems are said to be *unsolvable*. (See 13.4.2 for example)
  - Problems can be intractable if they would take too long or take too much memory to be of practical value. Example: chess



Answering: What can be computed?

- Experimentation
  - Some problems are too complex for analysis.
  - Implement a system and then study its behaviour.

But what will *I* do?

- Generally, as a Computer Science graduate you will

Write computer programs

Drink soda and eat pizza… a lot

Work with u

Solve problems!

clients to design programs

Have fun!

Get paid… well!

Lead other computer scientists

Test computer programs

Hardware Basics : CPU

- The *central processing unit* (CPU) is the “brain” of a computer.
  - The CPU carries out all the basic operations on the data.
  - Examples: simple arithmetic operations, testing to see if two numbers are equal.

Hardware Basics : Memory

- Memory stores programs and data.
  - CPU can only directly access information stored in *main memory* (RAM or Random Access Memory).
  - Main memory is fast, but *volatile*, i.e. when the power is interrupted, the contents of memory are lost.
  - Secondary memory provides more permanent storage: magnetic (hard drive, floppy), optical (CD, DVD)

Hardware Basics: I/O

- Input devices
  - Information is passed to the computer through keyboards, mice, etc.
- Output devices
  - Processed information is presented to the user through the monitor, printer, etc.

Programming Languages

- *High-level* computer languages
  - Designed to be used and understood by humans (C, Ada, Python, Java, .Net, etc…)
- *Low-level* language
  - Computer hardware can only understand a very low level language known as *machine language (binary, assembly which directly converts to binary)*

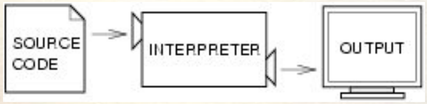
Programming Languages

- Low level version of “add two numbers”:
  - Load the number from memory location 2001 into the CPU
  - Load the number from memory location 2002 into the CPU
  - Add the two numbers in the CPU
  - Store the result into location 2003
- In reality, these low-level instructions are represented in binary (1’s and 0’s)

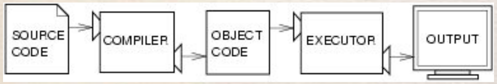
Conversion from high level to low level

- High-level language
  c = a + b
- This needs to be translated into machine language that the computer can execute.
- *Compilers* and *Interpreters* convert programs written in a high-level language into the machine language of some computer.

Compilers and Interpreters



Interpreted Language



Compiled Language

Interpreters

- *Interpreters* simulate a computer that understands a high-level language.
- The source program is not translated into machine language all at once.
- An interpreter analyzes and translates the source code instruction by instruction.

Python has both options, but we'll use it in an interpreted way in CS112!

The Magic of Python

IDLE is a programming environment for Python. Starting it you will see something like:

```
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.0.2
>>>
```

Python Prompt

- The ">>>" is a Python *prompt* indicating that Python is ready for us to give it a command. These commands are called *statements*.
-
  ```
  >>> print "Hello, world"
  Hello, world
  >>> print 2+3
  5
  >>> print "2+3=", 2+3
  2+3= 5
  >>>
  ```

Defining a Python Function

- Usually we want to execute several statements together that solve a common problem. One way to do this is to use a *function*.
-
  ```
  >>> def hello():
      print "Hello"
      print "Computers are Fun"


  >>>
  ```

Defining a Python Function

-
  ```
  >>> def hello():
      print "Hello"
      print "Computers are Fun"


  >>>
  ```
- The first line tells Python we are *defining* a new function called hello.
- The following lines are indented to show that they are part of the hello function.
- The blank line (hit enter twice) lets Python know the definition is finished.

Invoking a Function

-
  ```
  >>> def hello():
      print "Hello"
      print "Computers are Fun"


  >>>
  ```
- Notice that nothing has happened yet! We've defined the function, but we haven't told Python to perform the function!
- A function is *invoked* by typing its name.

-
  ```
  >>> hello()
  Hello
  Computers are Fun
  >>>
  ```

Parameters

- What's the deal with the ()'s?
- Commands can have changeable parts called *parameters* that are placed between the ()'s.

-
  ```
  >>> def greet(person):
      print "Hello",person
      print "How are you?”


  >>>
  ```

Parameters Example

-
  ```
  >>> greet("Terry")
  Hello Terry
  How are you?
  >>> greet("Paula")
  Hello Paula
  How are you?
  >>>
  ```
- When we use parameters, we can customize the output of our function.

Python Notes

- When we exit the Python prompt, the functions we’ve defined cease to exist!
- Programs are usually composed of functions, *modules*, or *scripts* that are saved on disk so that they can be used again and again.
- A *module file* is a text file created in text editing software (saved as “plain text”) that contains function definitions.
- A *programming environment* is designed to help programmers write programs and usually includes automatic indenting, highlighting, etc.

Complete Python Program

```
# File: chaos.py
# A simple program illustrating chaotic behavior

def main():
    print "This program illustrates a chaotic function"
    x = input("Enter a number between 0 and 1: ")
    for i in range(10):
        x = 3.9 * x * (1 - x)
        print x

main()
```

- We’ll use *filename.py* when we save our work to indicate it’s a Python program.
- In this code we’re defining a new function called **main**.
- The main() at the end tells Python to run the code.

Chaos output

```
>>>
This program illustrates a chaotic function
Enter a number between 0 and 1: .5
0.975
0.0950625
0.335499922266
0.869464925259
0.442633109113
0.962165255337
0.141972779362
0.4750843862
0.972578927537
0.104009713267
>>>
```

Comments

```
# File: chaos.py
# A simple program illustrating chaotic behavior
```

- Lines that start with # are called *comments*
- Intended for human readers and ignored by Python
- Python skips text from # to end of line

Inside a Python Program

```
def main():
```

- Beginning of the definition of a function called *main*
- Since our program has only this one module, it could have been written without the *main* function.
- The use of *main* is customary, however.

Python Print

```
print "This program illustrates a chaotic function"
```

- This line causes Python to print a message introducing the program.

Python Variable

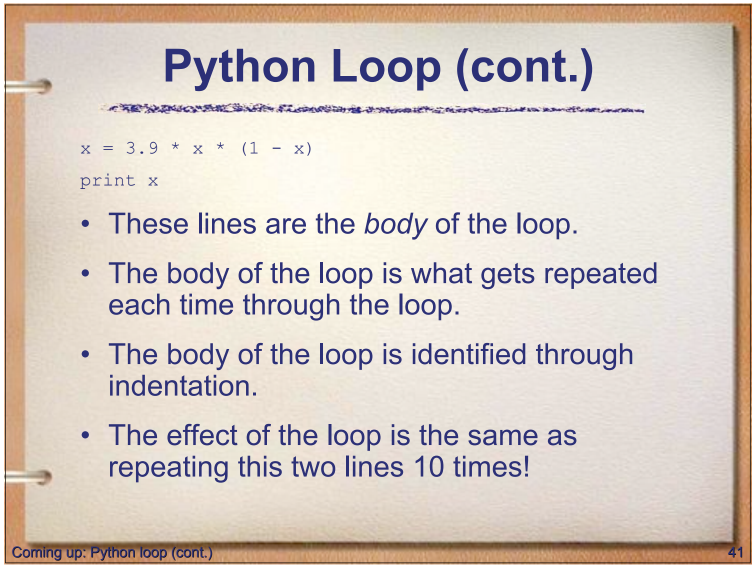
```
x = input("Enter a number between 0 and 1: ")
```

- x is an example of a *variable*
- A variable is used to assign a name to a value so that we can refer to it later.
- The quoted information is displayed, and whatever the user types in response is stored in x.

Python for loop

```
for i in range(10):
```

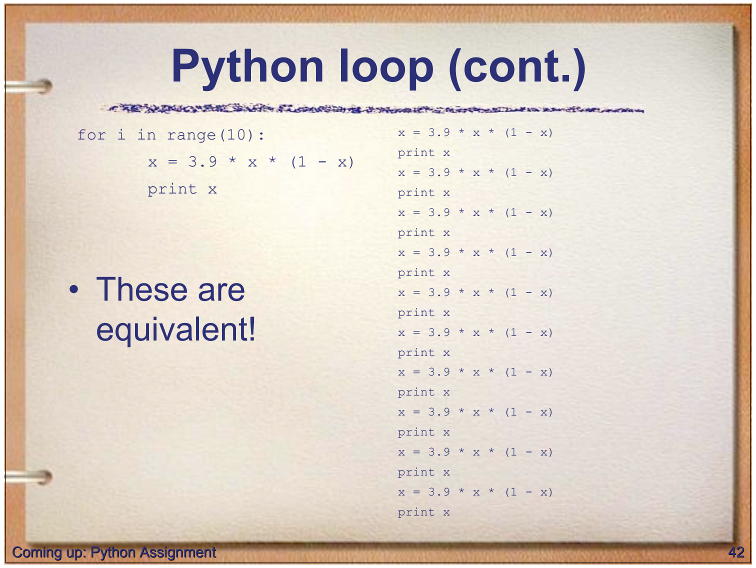
- For is a *loop* construct
- A loop tells Python to repeat the same thing over and over.
- In this example, the following code will be repeated 10 times.



Python Loop (cont.)

```
x = 3.9 * x * (1 - x)
print x
```

- These lines are the *body* of the loop.
- The body of the loop is what gets repeated each time through the loop.
- The body of the loop is identified through indentation.
- The effect of the loop is the same as repeating this two lines 10 times!

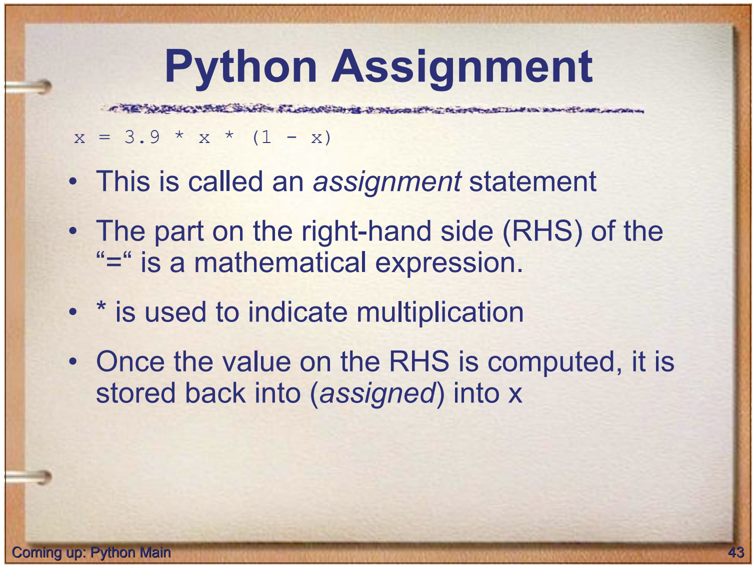


Python loop (cont.)

```
for i in range(10):
    x = 3.9 * x * (1 - x)
    print x
```

- These are equivalent!

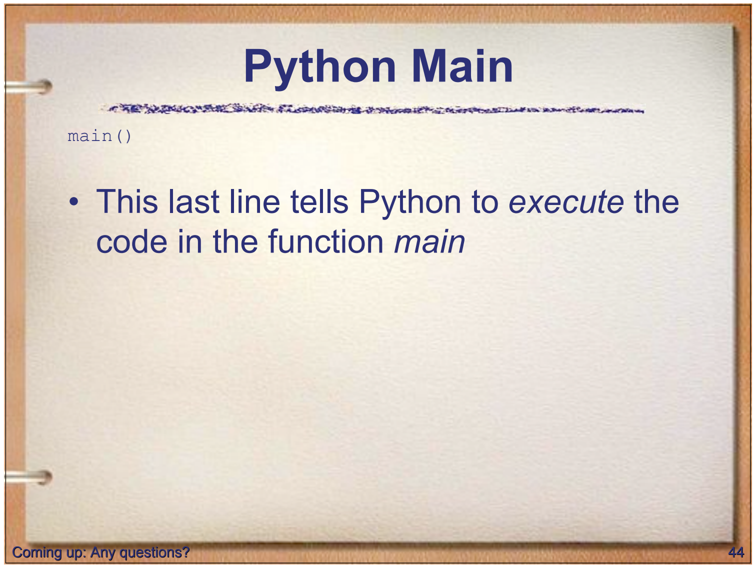
```
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
x = 3.9 * x * (1 - x)
print x
```



Python Assignment

```
x = 3.9 * x * (1 - x)
```

- This is called an *assignment* statement
- The part on the right-hand side (RHS) of the “=“ is a mathematical expression.
- * is used to indicate multiplication
- Once the value on the RHS is computed, it is stored back into (*assigned*) into x



Python Main

```
main()
```

- This last line tells Python to *execute* the code in the function *main*

Any questions?

References

- http://openbookproject.net//thinkCSpy/chap01.html