

Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.
<http://creativecommons.org/licenses/by/3.0/>.

Copyright © 2009, Charles Severance.

You assume all responsibility for use and potential liability associated with any use of the material. Material contains copyrighted content, used in accordance with U.S. law. Copyright holders of content included in this material should contact open.michigan@umich.edu with any questions, corrections, or clarifications regarding the use of content. The Regents of the University of Michigan do not license the use of third party content posted to this site unless such a license is specifically granted in connection with particular content. Users of content are responsible for their compliance with applicable law. Mention of specific products in this material solely represents the opinion of the speaker and does not represent an endorsement by the University of Michigan. For more information about how to cite these materials visit <http://michigan.educommons.net/about/terms-of-use>.

Any medical information in this material is intended to inform and educate and is not a tool for self-diagnosis or a replacement for medical evaluation, advice, diagnosis or treatment by a healthcare professional. You should speak to your physician or make an appointment to be seen if you have questions or concerns about this information or your medical condition. Viewer discretion is advised: Material may contain medical images that may be disturbing to some viewers.

Computers and Programs

Zelle - Chapter 1

Charles Severance - www.dr-chuck.com

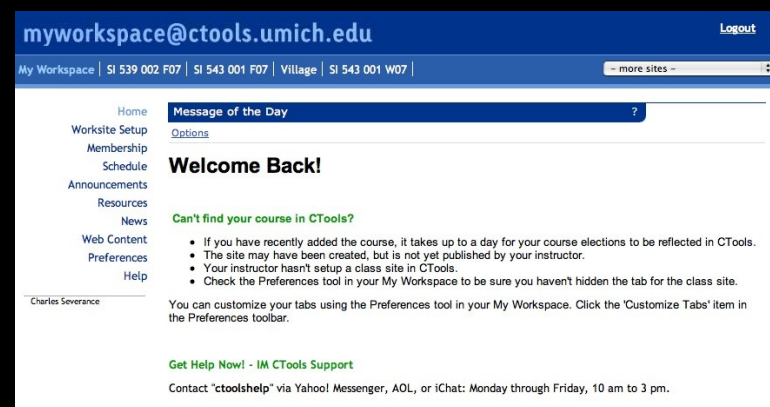
Users .vs. Programmers

- Users see computers as a set of tools - word processor, spreadsheet
- Programmers have some tools that allow them to build new tools
- Programmers sometimes write tools for lots of users and sometimes programmers write little widgets for themselves to automate a task

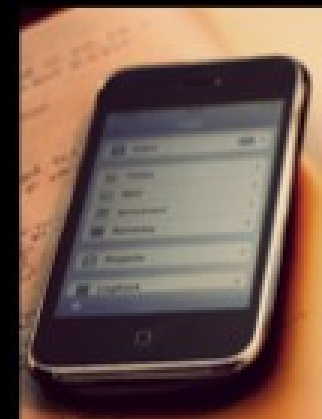
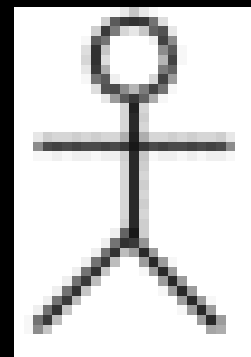
Why do we program?

- To get some task done - part of some non-programming job
 - Clean up survey data
- To produce something for others to use - a real programming job
 - Fix a performance problem in the Sakai software

(Screenshot) Source: ctools.umich.edu
(iPhone) CC BY: Johan Larsson (flickr)
<http://creativecommons.org/licenses/by/2.0>



User



Creators

Data

Information

....

Networks

From a software creator's point of view, we build the software. The end users (stakeholders/actors) are our masters - who we want to please - often they pay us money when they are pleased. But the data, information, and networks are our problem to solve on their behalf. The hardware and software are our friends and allies in this quest.

What is Code? Software? A Program?

- A set of stored instructions
 - It is a little piece of our intelligence in the computer
 - It is a little piece of our intelligence we can give to others - we figure something out and then we encode it and then give it to someone else to save them the time and energy of figuring it out
- A piece of performance art

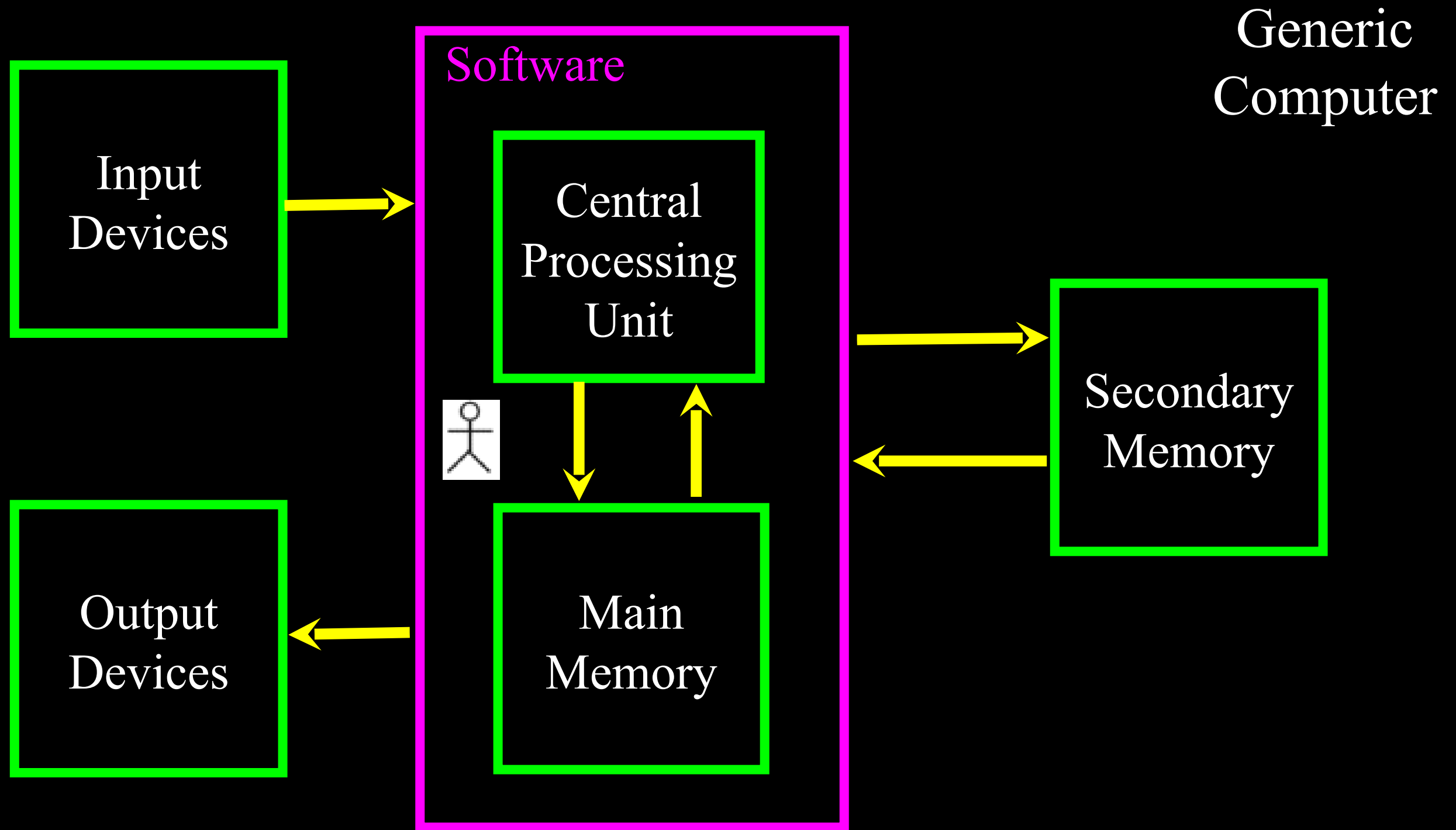
How Many Lines in a File?

- You could read and understand
 - Chapter 4 page 110
 - Chapter 11 Page 341
 - Appendix A Page 447

Or I could send you this in E-Mail:

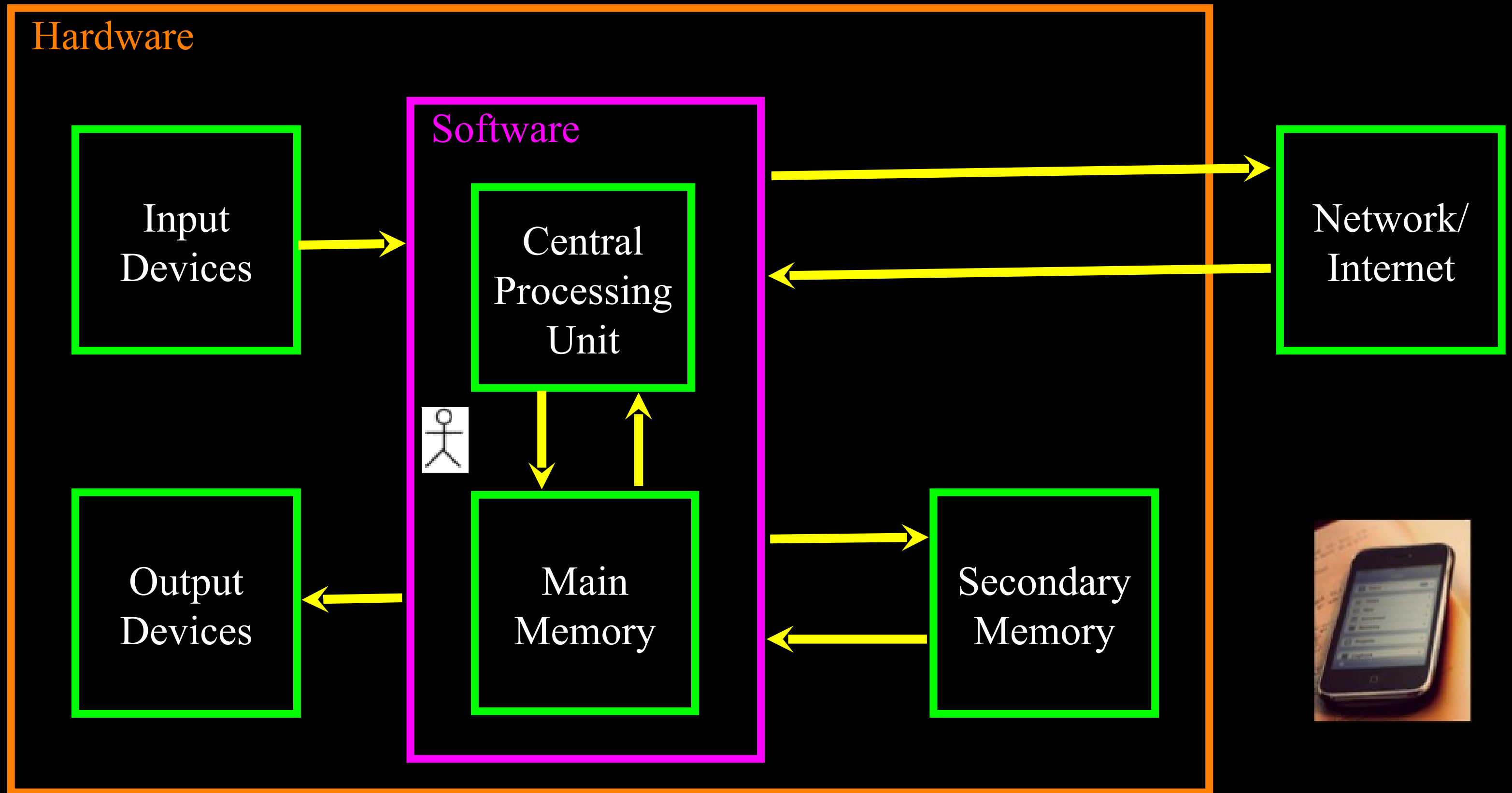
```
infile = open("mbox", "r")  
print len(infile.readlines())
```

Hardware Architecture



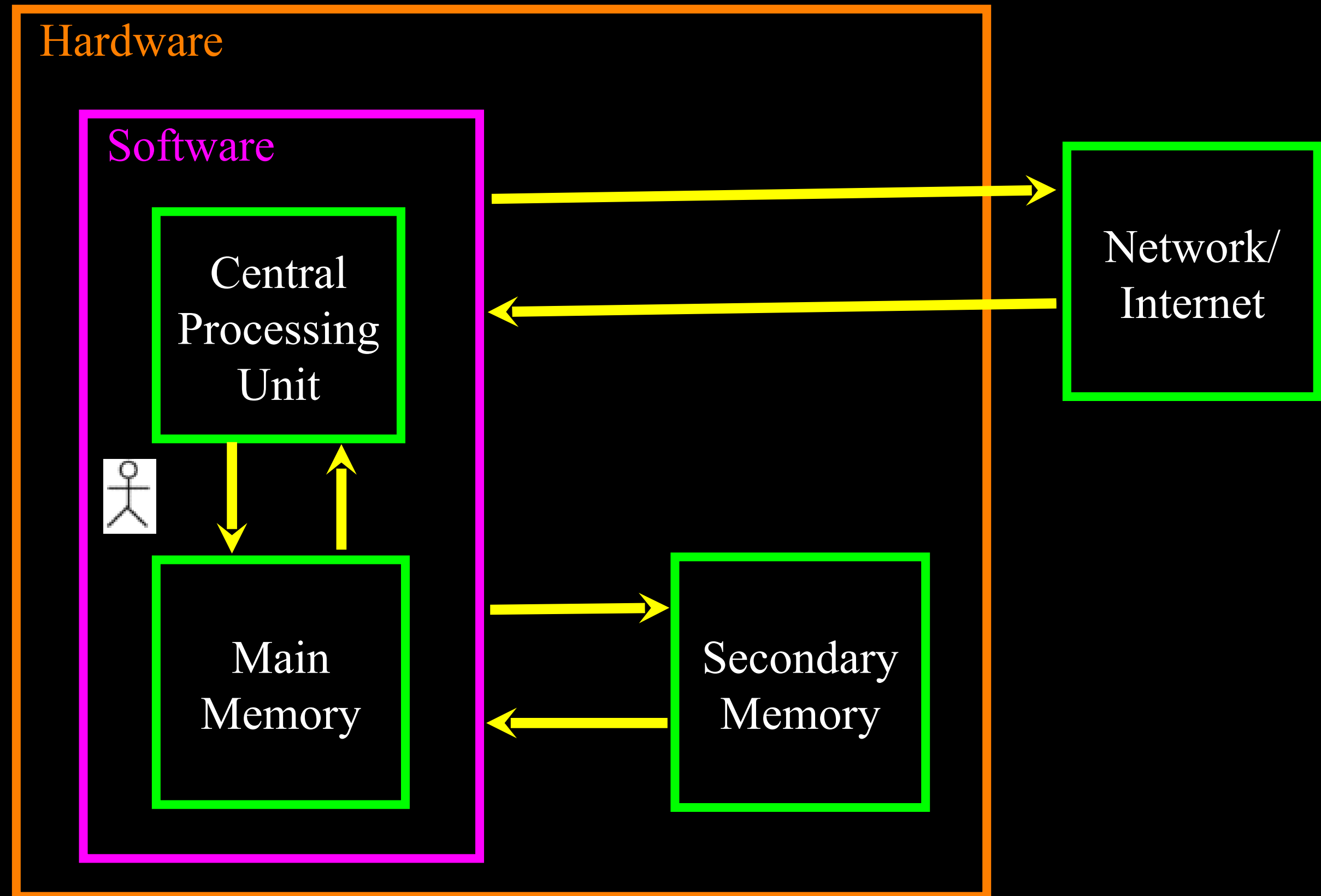
Definitions

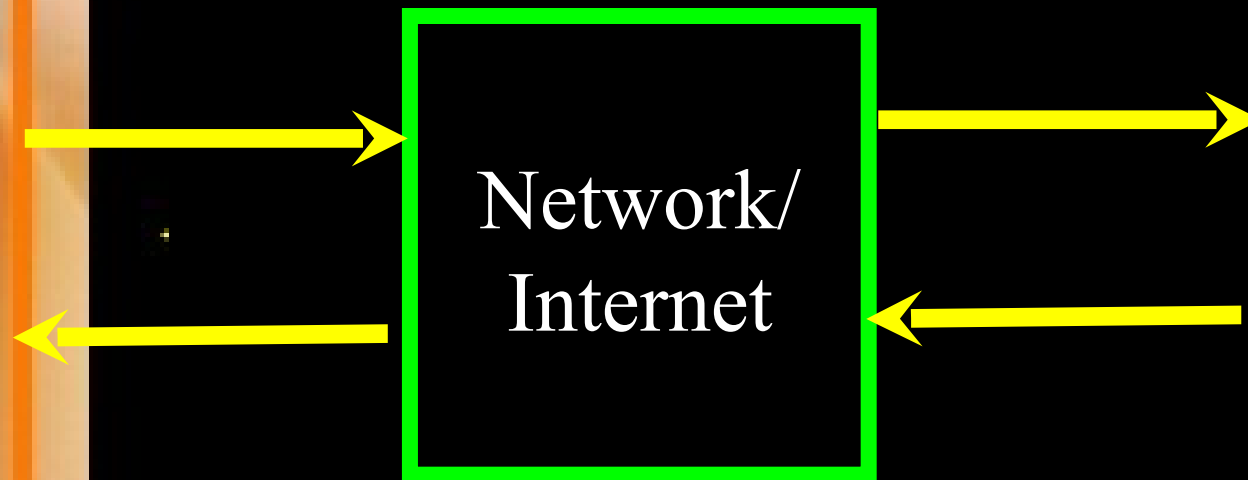
- Input Devices: Keyboard, Mouse, Touch Screen
- Output Devices: Screen, Speakers, Printer, DVD Burner
- Central Processing Unit: Runs the Program - AKA The CPU is always wondering “what to do next”? Not the brains exactly - very dumb but very very fast
- Main Memory: Fast small temporary storage - lost on reboot - aka RAM
- Secondary Memory: Slower large permanent storage - lasts until deleted - disk drive / memory stick



Web Server

A web server often functions with no input or output devices connected to the system. It takes incoming requests from the network - does some work with those requests and send output back across the network.





Hardware

Software

Programmer Tools

Becoming a Programmer

- We use the computer - we just have to learn some programmer tools
 - Compiler - Takes our code and makes it executable
 - Interpreter - Reads our code and runs it
 - Development Environment - Helps us write code

When a Program Runs...

- When a program runs it:
 - Takes some **input** data
 - **Processes** the data using a set of instructions (a program)
 - Produces some **output**
 - Think of it as “value add”



An example program takes a **text file** as its **input** and **counts the lines in the file** and **prints out the number of lines in the file.**

Programmer Tools

- We use the computer - we just have some new tools
 - Development Environment - A “Word Processor” or “Text Editor” for Programmers - we write code in a development environment
 - Compiler - Takes our code and makes an executable version of our program
 - Interpreter - Reads our code and runs it directly - Python is an interpreted language - Python is an interpreter

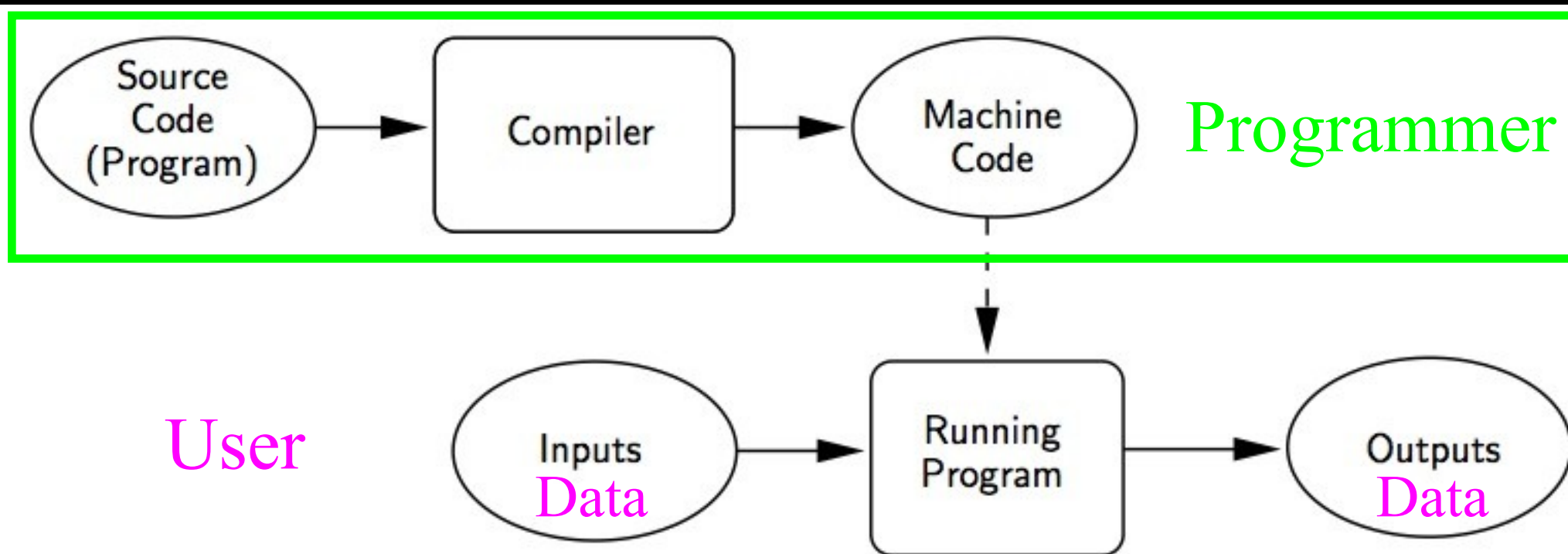


Figure 1.2: Compiling a High-Level Language

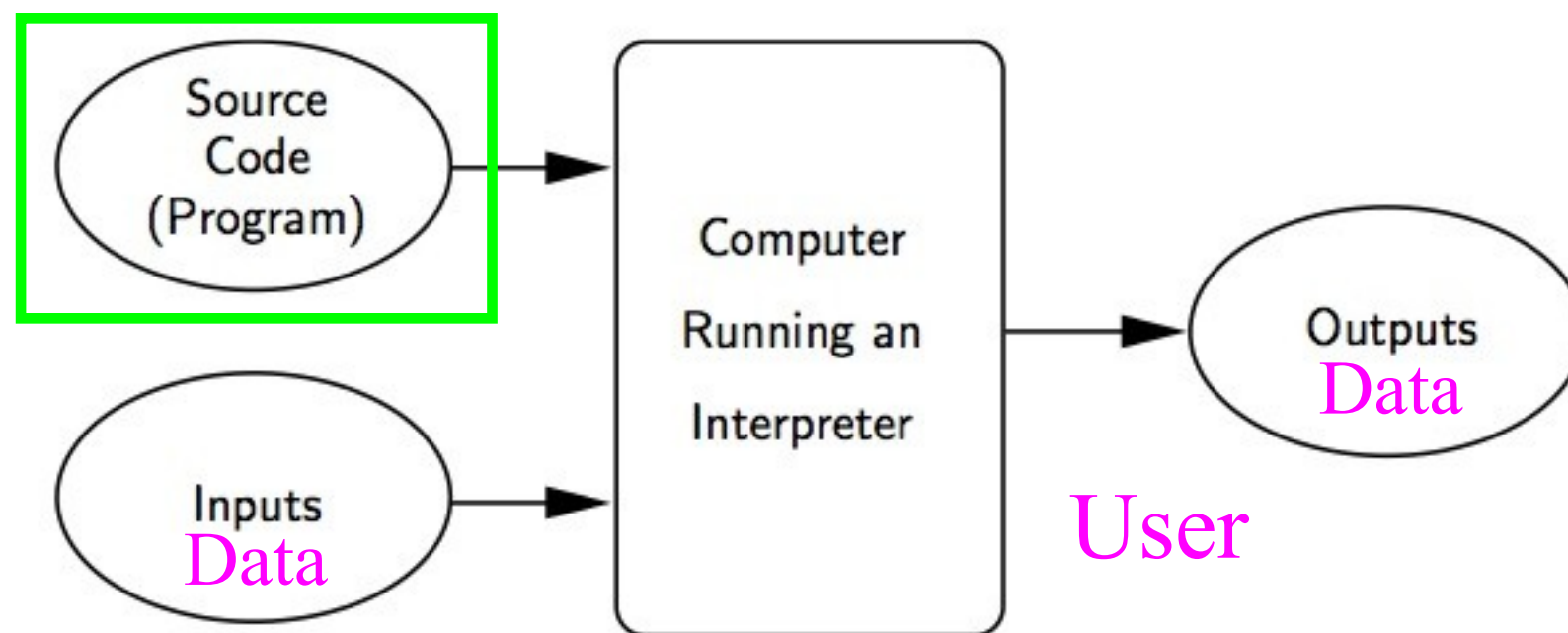


Figure 1.3: Interpreting a High-Level Language.

A programmer develops a program.

If a compiler is used the compiler translated the source to machine code for distribution.

If an interpreter is used, the programmer simply distributes the source code.

Terms

- Source code - the programs we humans write - and read - written in a programming language - source code is generally portable across systems
- Machine code - what really runs on the machine - not very readable - produced by a compiler - machine code is unique to hardware and operating system.



search

[Advanced Search](#)

Screen styles

normal large* userpref

» [Download](#)[ABOUT](#) >>[NEWS](#) >>[DOCUMENTATION](#) >>[DOWNLOAD](#) >>[Releases](#)[Windows, DOS](#)[Macintosh](#)[Linux](#)[Other](#)[Source](#)[Python 3000](#)[SVN Access](#)[Contributed Software](#)[COMMUNITY](#) >>[FOUNDATION](#) >>[CORE DEVELOPMENT](#) >>[LINKS](#) >>

Download Standard Python Software

Note: there's a [security fix](#) for Python 2.2, 2.3 and 2.4. Of the releases below, only 2.4.4 and 2.5 and later include the fix.

[The current production version](#) is Python 2.5.2. You should start here if you want to learn Python or if you want the most stable version. Here are some quick download links. For the MD5 checksums and OpenPGP signatures, look at the [detailed Python 2.5.2](#) page:

- [Python 2.5.2 compressed source tarball](#) (for Unix or OS X compile)
- [Python 2.5.2 bziped source tarball](#) (for Unix or OS X compile, more compressed)
- [Python 2.5.2 Windows installer](#)

(Windows binary -- does not include source)

- [Python 2.5.2 Windows AMD64 installer](#) (Windows AMD64 binary -- does not include source)
- [Python 2.5.2 Windows Itanium installer](#) (Windows Itanium binary -- does not include source)

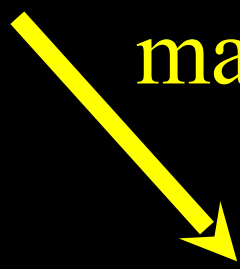
Your Source Code

```
main() {  
    printf("Hello world\n");  
}
```



```
^@^@^@^@__DATA^@^@^@^@^@^@^@^@^@^@  
^@^@^@0^@^@^@^@^@q^@^@  
^@^@^@^@^@B^@^@^@^@^@^@^@^@^@^@  
^@^@^@^@^@^@^@^@^@^@__nl_symbol_ptr  
r^@__DATA^@^@^@^@^@^@^@^@^@^@^@^@  
@0t^@^@^@^@P^@^@  
t^@^@^@^@B^@^@^@^@^@^@^@^@^@^@^@^@  
F^@^@^@^@Q^@^@^@^@^@__la_symbol_ptr  
^@__DATA^@^@^@^@^@^@^@^@^@^@^@^@  
0<84>^@^@^@D^@^@^@A^@^@/usr/lib/  
libmx.A.dylib^@^@^@^@^@L^@^@^@4  
^@^@^@XC^@m^@X^A^C^@A^@^@/usr/  
lib/libSy
```

The C Language
Compiler Reads
your source code
and produces your
machine code.



```
^?  
ELF^A^A^A^@^@^@^@^@^@^@^@^@^@^@^@B^@C^@A^@^@^@\xa0\x82^D^H4^@^@^@\x90^] ^@^@^@^@^@^@4^  
@ ^@G^@(^@$^@!  
^@F^@^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@E^@^@^@D^@^@^@C^@^@^@T^  
A^@^@^@T\x81^D^H^T\x81^D^H^S^@^@^@S^@^@^@D^@^@^@A^@^@^@A^@^@^@D^HQVhT\x83^D^H\xe8\x  
b7\xff\xff\xff\xff4\x90\x90U\x89\xe5S\xe8^@^@^@^@[\x81\xc3_^R^@^@P\x8b\x83\xff\xff\  
xff\xff\x85\xc0t^B\xff\xdb]\xfc\xc9\xc3\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90  
0U\x89\xe5\x83\xec^H\x80=L\x95^D^H^@t^L\xeb^|\x83\xc0^D\xa3H\x95^D^H\xff\xdb2\xa1H\  
x95^D^H\x8b^P\x85\xdb2u\xeb\xc6^EL\x95^D^H^A\xc9\xc3\x90U\x89\xe5\x83\xec^H\xa1|\x9  
4^D^H\x85\xc0t!\xb8^@^@^@\x85\xc0t^X\xc7^D$|\x94^D^H\xe8\xbc|\xfb\xf7\x8d\xb6^@^  
^@^@^@\x8d\xbf^@^@^@\xc9\xc3\x90\x90U\x89\xe5\x
```

Your Machine Code

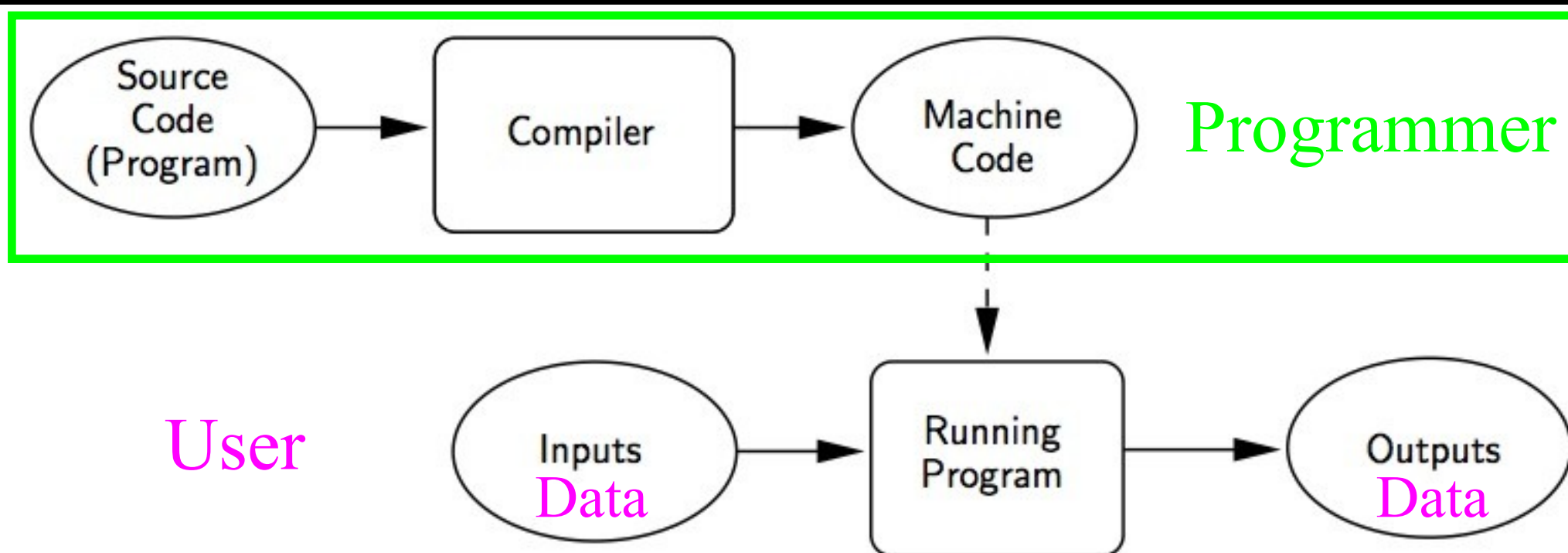


Figure 1.2: Compiling a High-Level Language

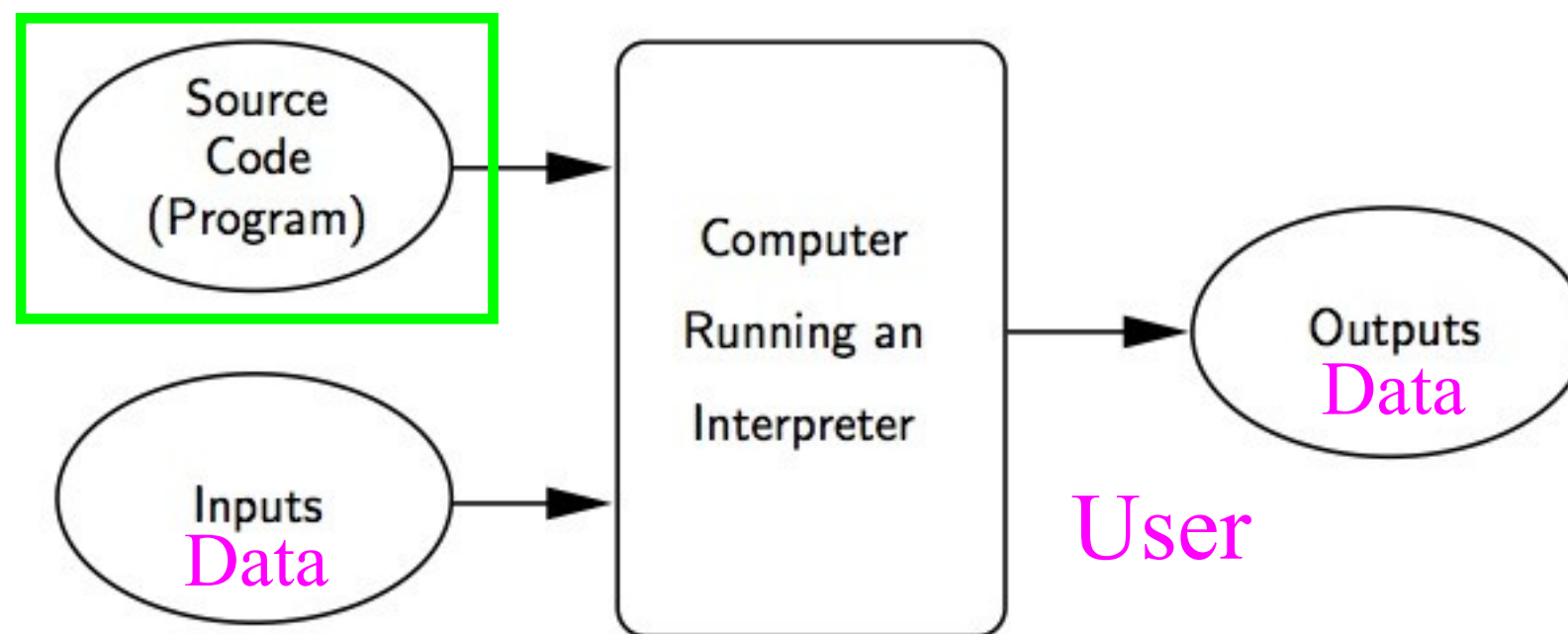


Figure 1.3: Interpreting a High-Level Language.

A programmer develops a program.

If a compiler is used the compiler translated the source to machine code for distribution.

If an interpreter is used, the programmer simply distributes the source code.

Compiler .vs. Interpreter

- Only the programmer needs to have the compiler - once the compiler is done - the executable program is self-contained
- The programmer keeps the source code and distributes the executable - different executables are needed for Mac, PC, etc.
- Both the programmer and user need to have the Interpreter installed on their system
- Generally the programmer distributes the source code of the program

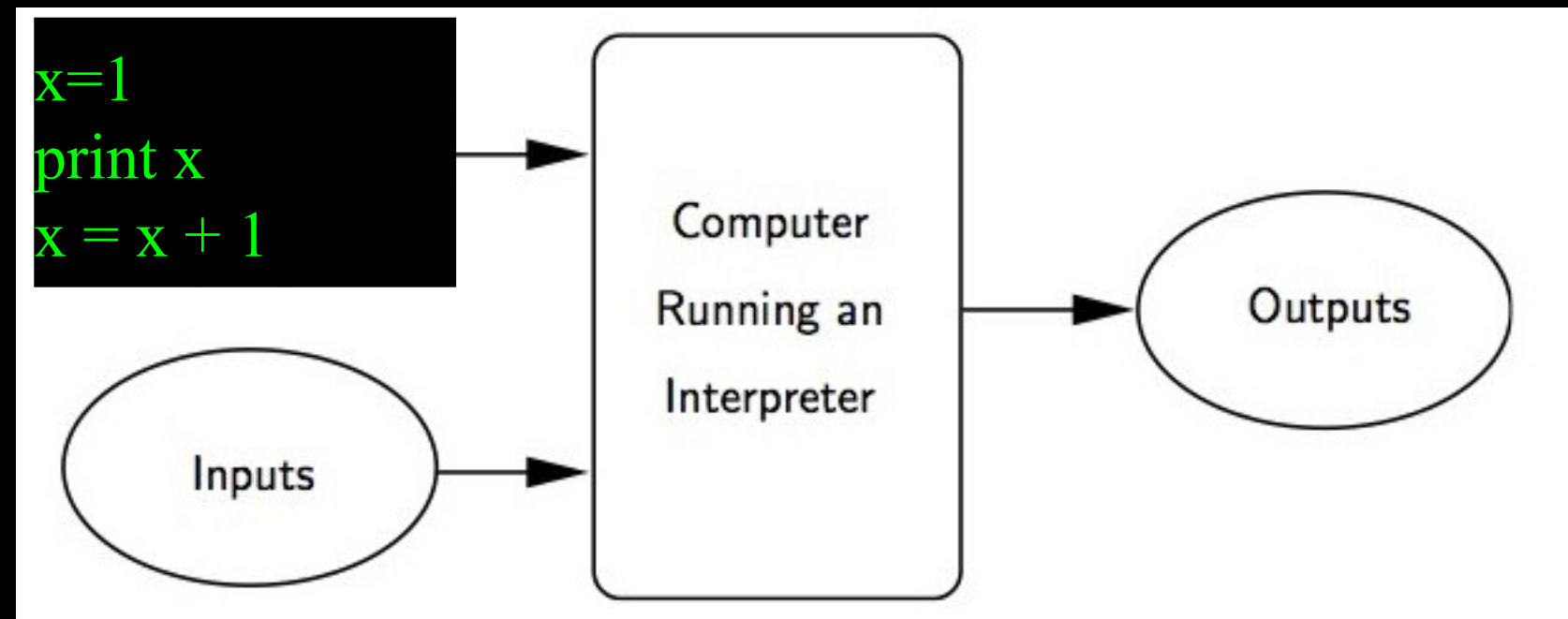
Python is an Interpreter

- To run Python programs, users must install Python on their computers
- Development is quick and easy - we simply make a change to our program and run it again in a single step
- For data analysis - Python is just a tool that you keep on your desktop or laptop
- Interpreters are more convenient when the user and programmer are the same person

Running Python Interactively

Python Interactive

- Since Python is interpreted we can just type programs directly into Python
- See Also <http://datamech.com/devan/trypython/trypython.py>



```
csev$ python
```

```
Python 2.5 (r25:51918, Sep 19 2006, 08:49:13)
```

```
[GCC 4.0.1 (Apple Computer, Inc. build 5341)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> x = 1
```

```
>>> print x
```

```
1
```

```
>>> x = x + 1
```

```
>>> print x
```

```
2
```

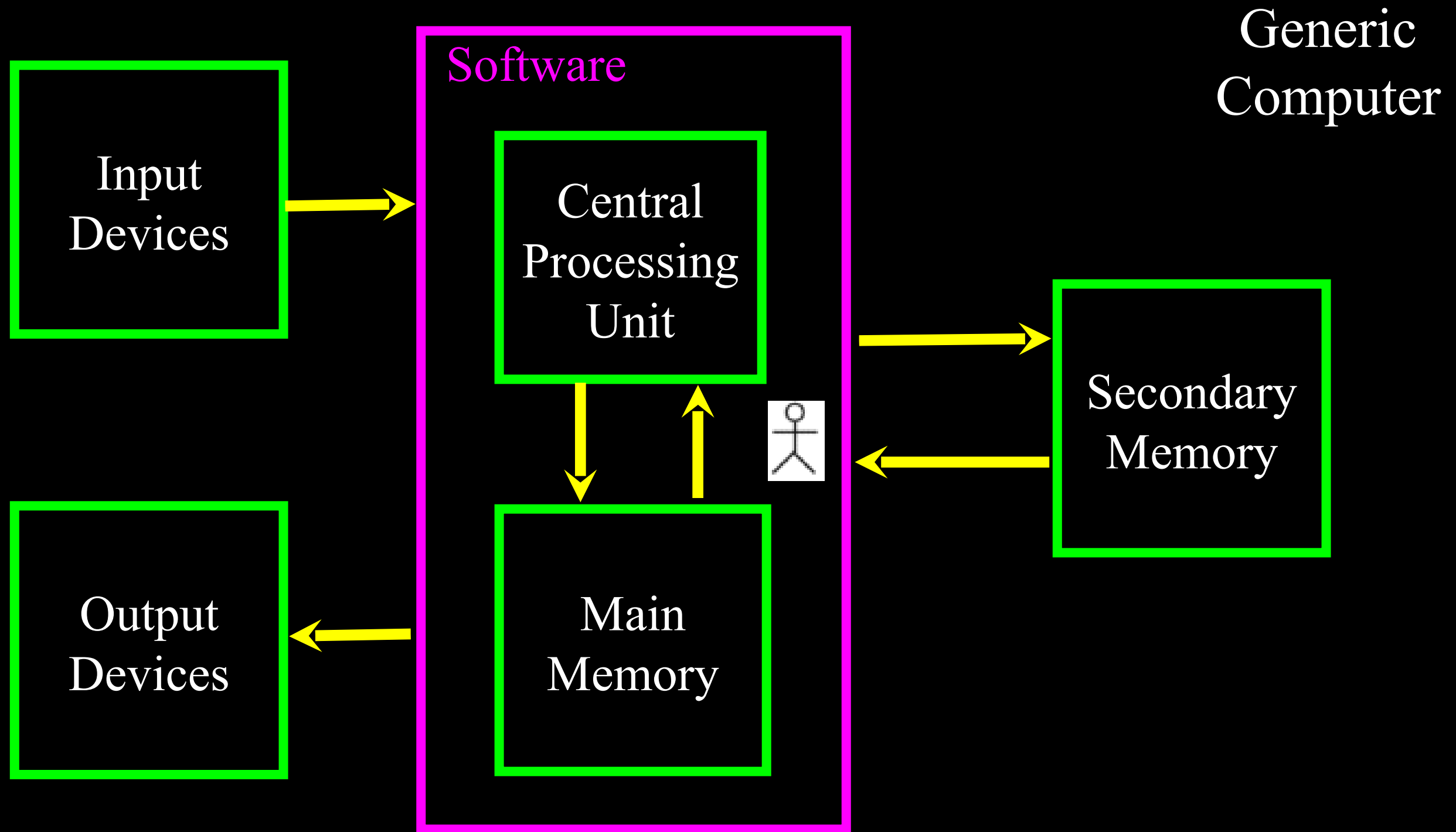
```
>>> exit()
```

This is a good test to make sure that you have python correctly installed.

Syntax Errors

- The computer has a language where you can tell it what you want to do - this is Python
- It seems unfair when you submit a program to the computer and it says “syntax error” - given that it **knows** the language and you are just learning it. It seems rude and cruel.
- You must remember that you are intelligent and **can** learn - the computer is simple and very fast - but cannot learn - so it is easier for you to learn Python than for the computer to learn English...

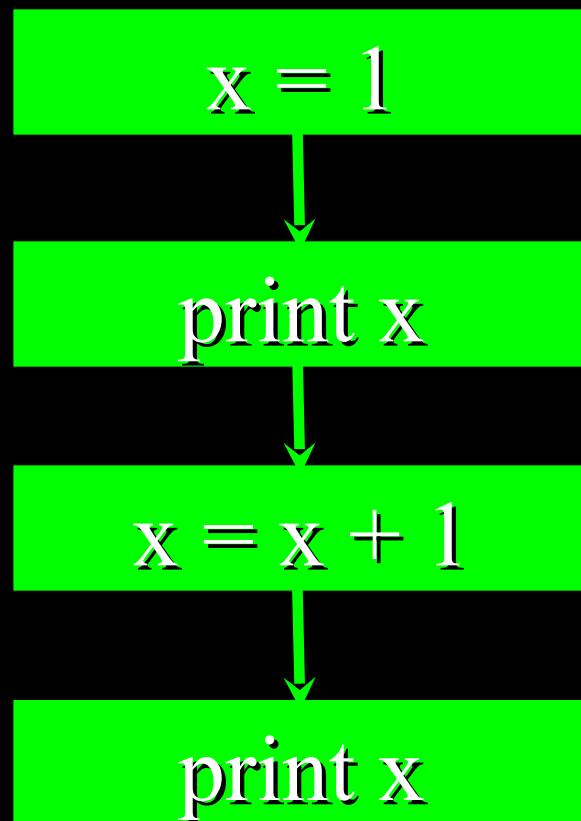
The Essence of Programming



Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order
- Some steps are conditional - they may be skipped
- Sometimes a step or group of steps are to be repeated
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program

Sequential Steps



Program:

`x = 1`

`print x`

`x = x + 1`

`print x`

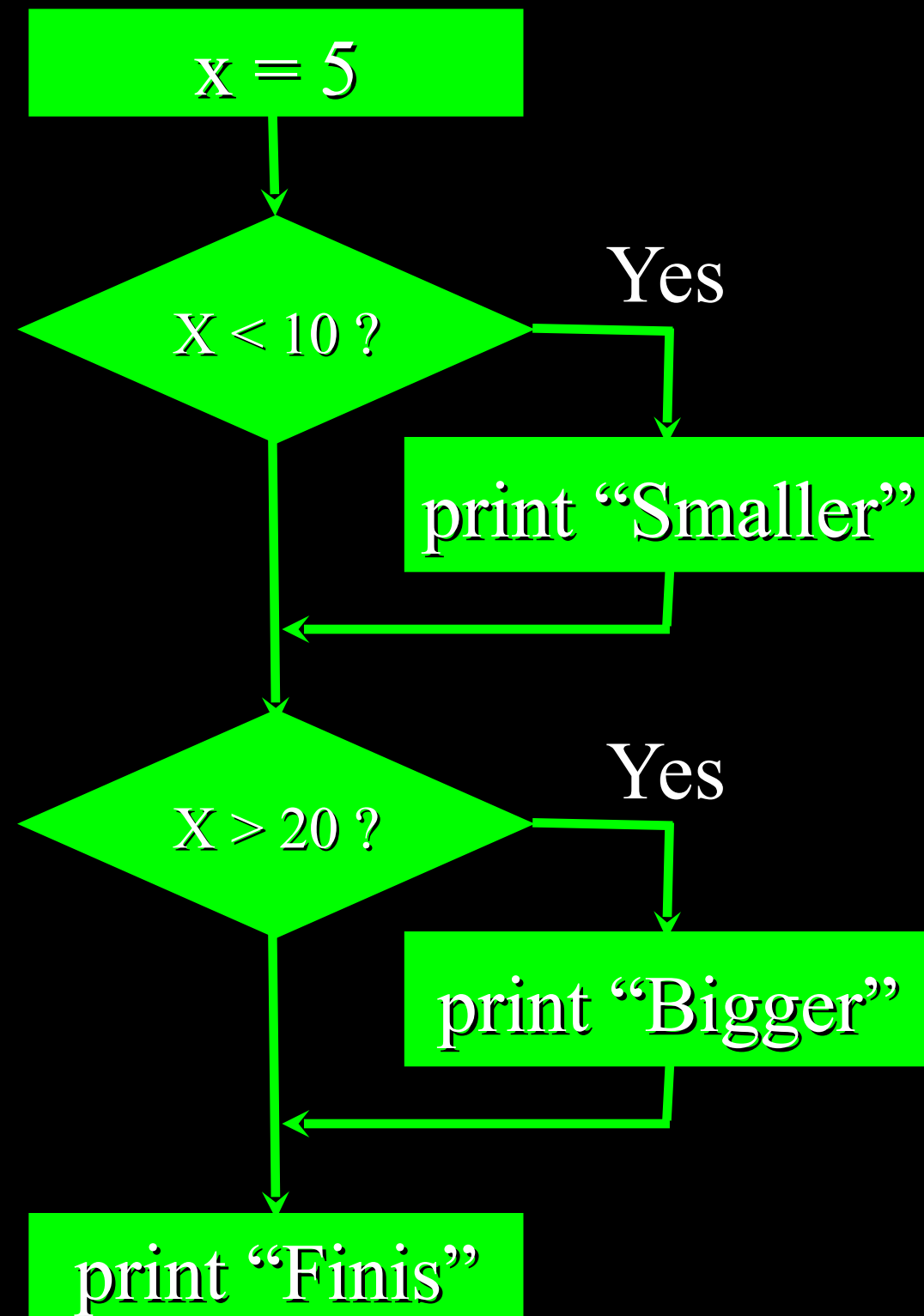
Output:

1

2

When a program is running, it flows from one step to the next.
We as programmers set up “paths” for the program to follow.

Conditional Steps



Program:

```
x = 5
```

```
if x < 10:
```

```
    print "Smaller"
```

```
if x > 20:
```

```
    print "Bigger"
```

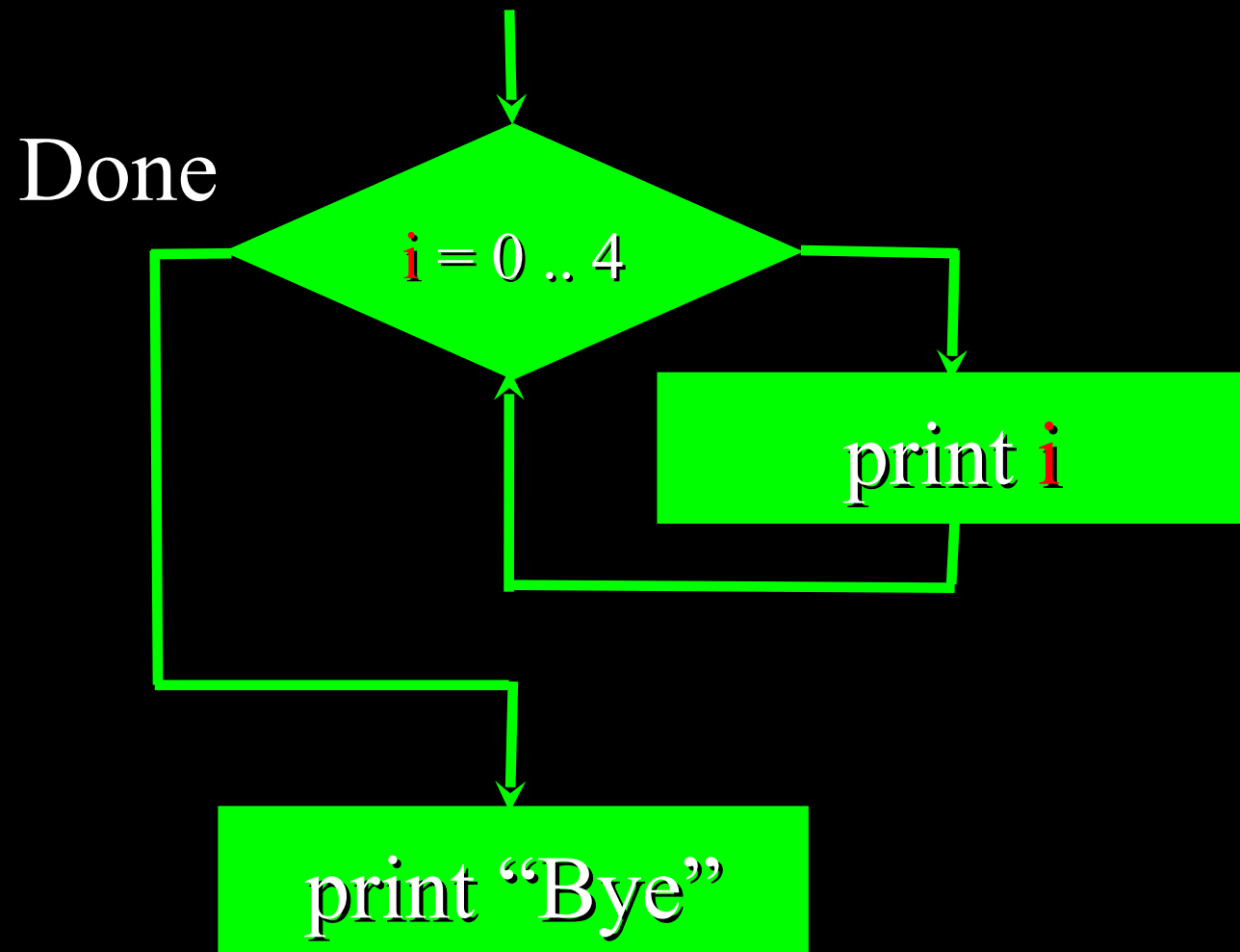
```
print "Finis"
```

Output:

Smaller

Finis

Repeated Steps



Program:

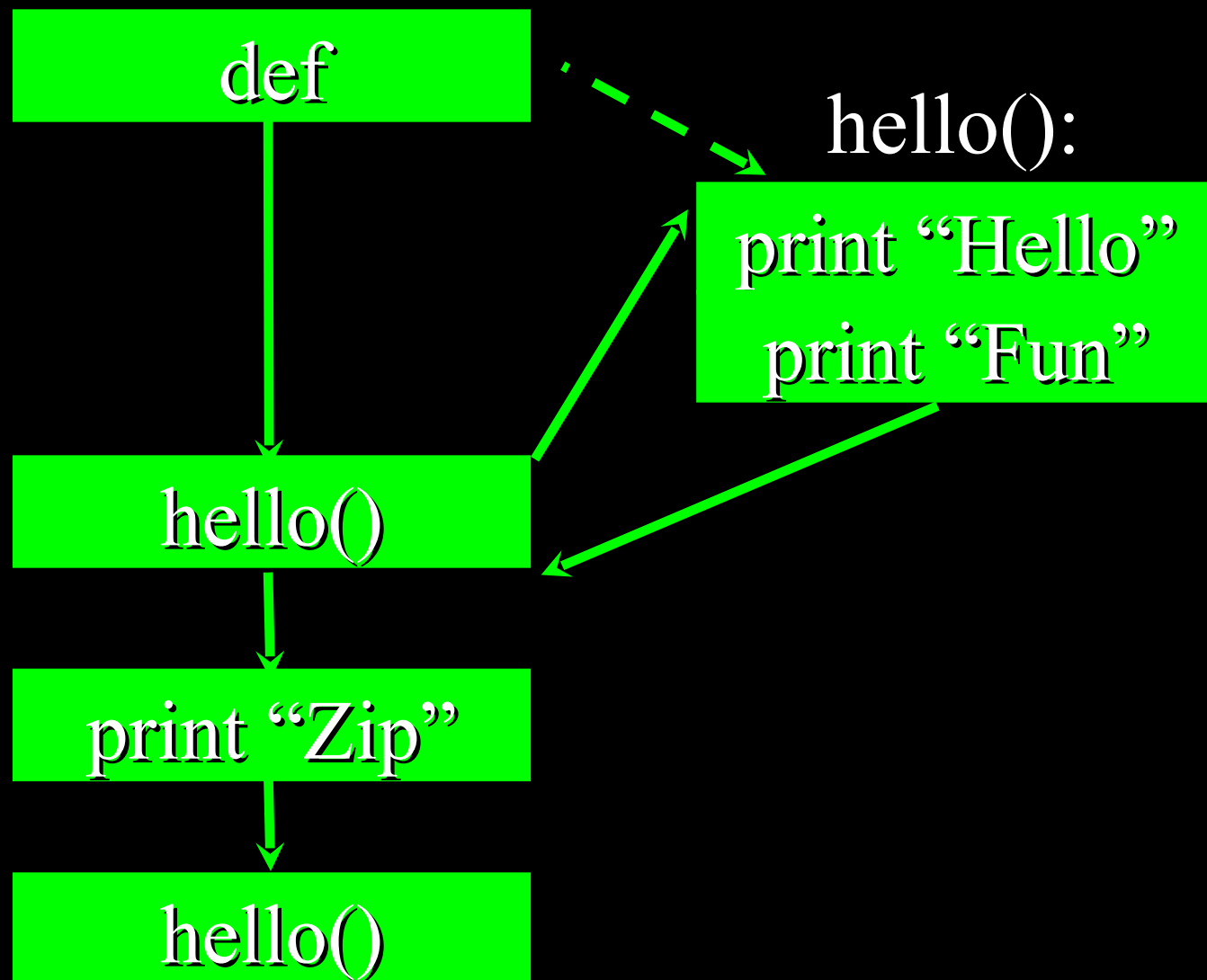
```
for i in range(5) :  
    print i  
  
print "Bye"
```

Output:

0
1
2
3
4
Bye

Loops (repeated steps) have **iteration variables** that change each time through a loop. Often these **iteration variables** go through a sequence of numbers.

Stored (and reused) Steps



Program:

```
def hello():  
    print "Hello"  
    print "Fun"
```

```
hello()  
print "Zip"  
"hello()
```

Output:

```
Hello  
Fun
```

```
Zip  
Hello  
Fun
```

We call these little stored chunks of code “subprograms” or “functions”.

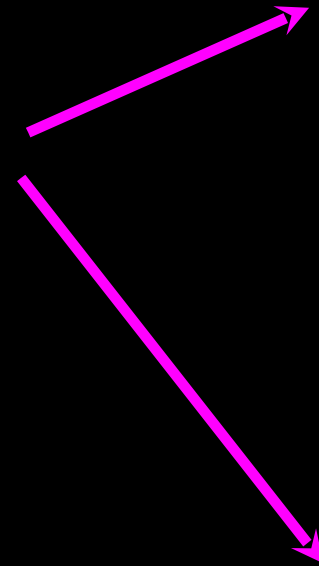
A Python Program

```
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()
```

```
$ python chaos.py This program illustrates a
chaotic function Enter a number between 0
and 1: 0.6
0.936
0.2336256
0.698274248196
0.821680557759
0.571434313164
0.955098841721
0.16725167263
0.543186347468
0.96772626363
0.121805355011
```

Stored steps



```
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()
```

Calling the stored steps

Output

Input

```
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()
```


Repeated Code

```
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()
```

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

```
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()
```



The colon (:) starts a block of indented code

Indented code continues until a line is encountered that is less indented.

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


Variables and Assignment Statements

Variables and Assignments

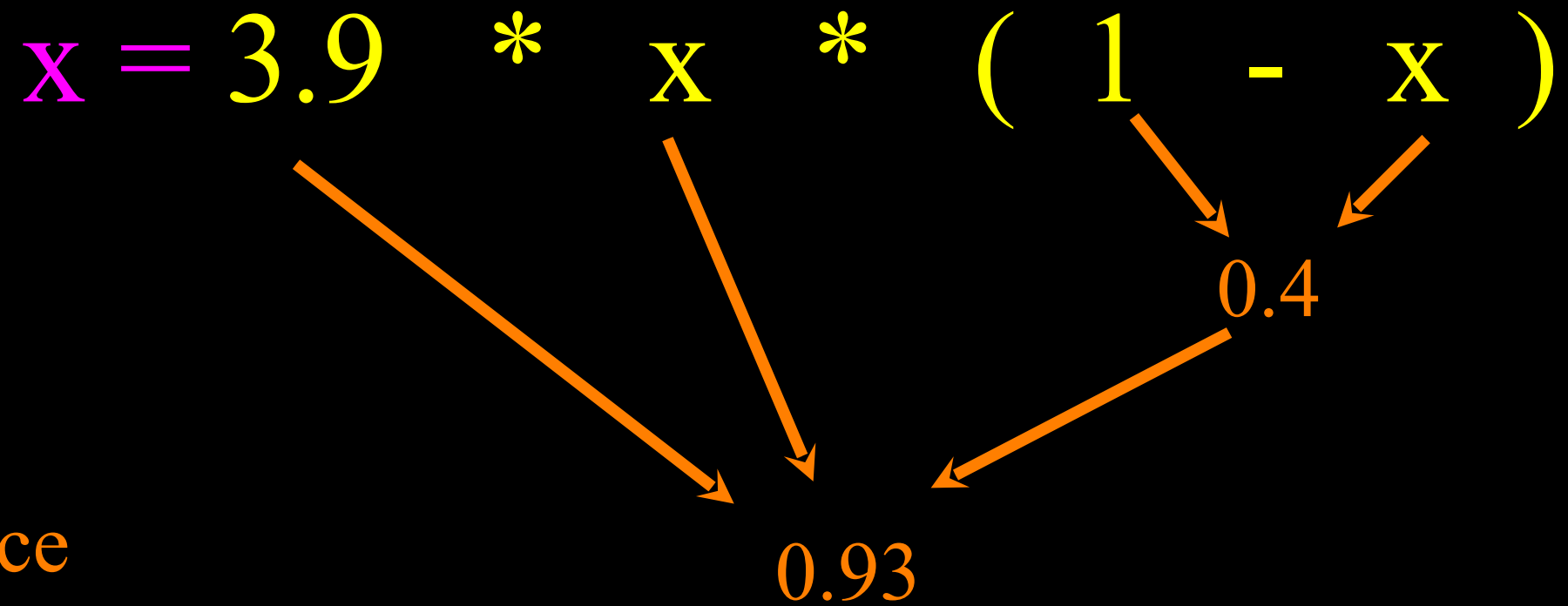
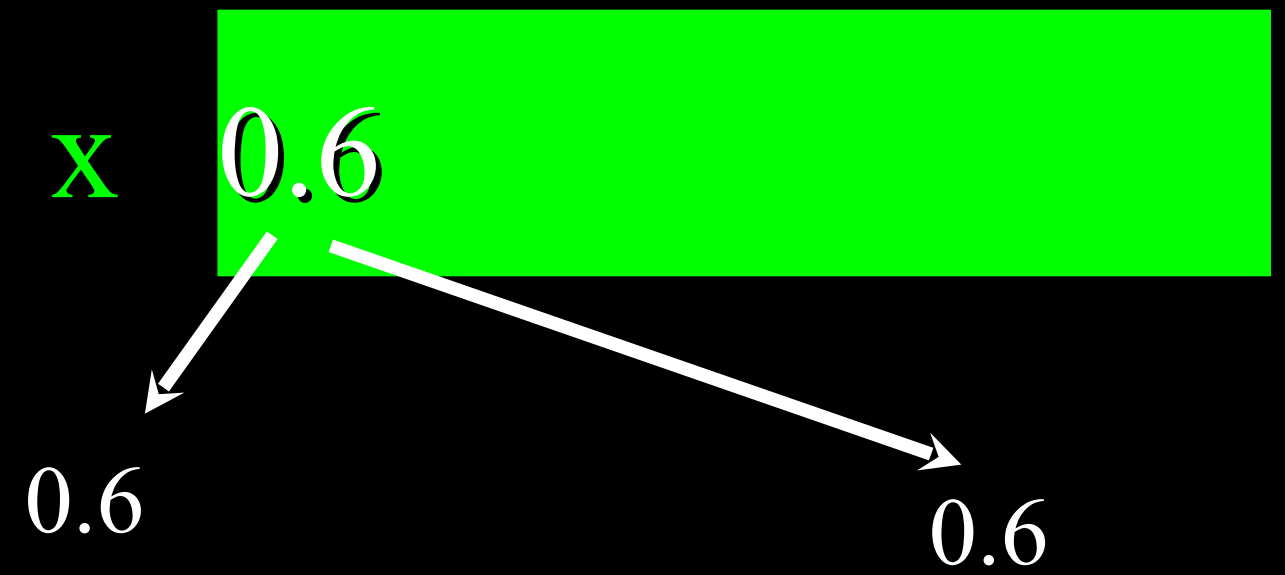
- A **variable** is a scratch local to store some value such as a number or a string
- An assignment statement consists of an **expression on the right hand side** and a **variable** to store the result

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

Assignment Statement into the variable named x

```
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()
```

A variable is a memory location used to store a value (0.6).



Left side is an expression. Once expression is evaluated, the result is placed in (assigned to) x .

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.93).



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

Right side is an expression. Once expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e. x).

0.93

Comments in Python

Comments in Python

- Anything after a # is ignored by Python
- Why comment?
 - Describe what is going to happen in a sequence of code
 - Document who wrote the code or other ancillary information
 - Turn off a line of code - perhaps temporarily

Comments can document our programs.

Comments can also be used to temporarily turn off lines of code without deleting those lines in case we want them back later.

```
# 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):
        # print i
        x = 3.9 * x * (1 - x)
        print x

main()
```

Summary

- This is a quick overview of Chapter 1
- We will revisit these concepts throughout the course
- Focus on the big picture