CS151 Fall 2012 Lecture 2

Stephanie R Taylor

Sep 7, 2012

1 Administrative Topics

- Stephanie takes her quiz.
- Did everyone make it to lab this week?
- Does everyone have a partner?
- I have no official office hours today, but will be in my office until 2:30.
- Plan for today: talk about algorithms, go through example code for project

2 Computers are Stupid

Computers are supremely stupid (they have a very limited set of instructions they are capable of following). To communicate with computers (i.e. get them to do something for us), we need to be totally unambiguous.

What is a computer? The computers we work with today have what is called the von Neumann architecture. There are several components (see Fig 1).

- Controller

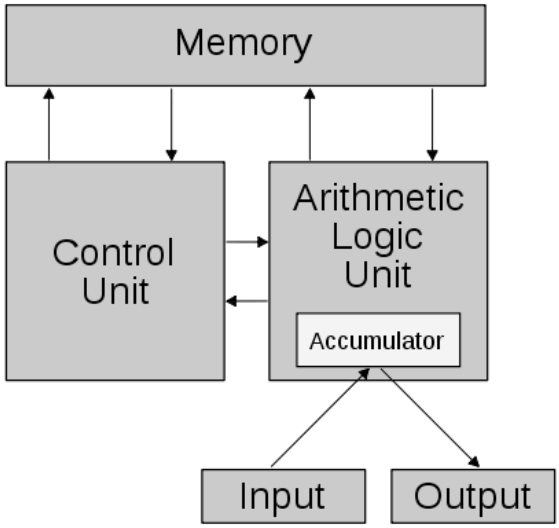

Figure 1: von Neumann Archicture

- Arithmethic Logic Unit which together make the Central Processing Unit, or CPU this is where the computation is done - e.g. where numbers are added (Note: the controller feeds instructions to the ALU.)
- Main Memory (Random Access Memory) this is the memory in your computer that is used only when the computer is on. For example, when you are browsing the web, the contents of the web page are stored in main memory. So is the information about what must be drawn to your screen.
- input, e.g. from your keyboard, mouse, or your hard drive. It is your hard drive where files are stored. Anything that is saved while the power is off is saved on the hard drive. Now we can see where my misunderstanding was in regards to the word processor I used on the Apple][e. "Load" meant "load from a disk into main memory".
- output, e.g. to your screen or to your hard drive

I said computers are stupid. Why? well the CPU has a limited set of instructions. It is moving data around without any concept of the "bigger picture".

What is this data that computers are moving around? at the physical level, there are wires with electrical signal. Hi voltage is considered to be a 1. Lo is considered to be 0. Each wire is a bit, and they are grouped as bytes (1 byte = 8 bits) The instructions the CPU carries either don't bother to interpret the data at all (i.e. it is just moving it), or interpret the data as numbers - hence the name "arithmetic logic unit".

How do we get the computer to do what we want it to do? Do we need to think about bits? To some extent, the answer is "yes", because we need to be conscious of what data is moving around the machine. However, we don't need to get bogged down into too much detail. We have programming languages which help us tell the computer what to do.

3 Humans are Smart

Humans are highly intelligent. Their communication is ambiguous and is open to interpretation.

We want to use computers. Computer scientists make them do many, varied things. Recall our list from Wednesday. What is common among these items? One unifying theme is "precise thinking". But precise thinking about what? About how to do things - how to network two computers, how to simulate biological systems. They all study *process*: how we do things, how we specify what we do, how we specify what the stuff is that we are processing.

4 Algorithms and Programs

This brings us to the term "algorithm". An algorithm is a sequence of instructions that performs a specific task or solves a specific problem.

An **algorithm** is a sequence of instructions to perform a specific task or to solve a specific problem. An algorithm should be unambiguous to a human, but it need not be executable by a computer.

Program – an implementation of an algorithm in a computer programming language, making it unambiguous to a computer

In your first project, you will be developing algorithms to draw specific shapes. You will then *implement* of your shape-drawing algorithms by writing Python code. Your algorithm is a precise set of instructions a human can understand, and your code will be a precise set of instructions that Python can understand.

5 Python

For the rest of the class, we will spend some time talking about how to write Python code for the projects.

You have already seen that Python can be run in interpreter mode, by simply typing python in the Terminal window. It can also be used to run specific Python programs, but typing python and then the name of the program file after it. That is what we will talk about in a minute, but let's talk about turtle first.

Our first few projects will use the turtle to draw pictures. To draw with the turtle, you *call*, or *execute*, the turtle functions. A function is a chunk (sorry, chunk is not a technical term) of code with a particular name and a particular purpose. In lab, you executed several turtle functions, including `left`, `forward`, and `reset`.

Python has a lot of packages (of which the turtle is one). In order to gain access to the functions in a particular package, or *module*, you need to import that module. So you typed `from turtle import *` in the interpreter.

[Demo turtle functions, pointing out the need for parentheses to make sure it is actually executed].

When you are writing programs, it is best to write them in files, rather than type them the interpreter. This way, you can write your once, then edit it as necessary, if you don't like your first attempt (which you NEVER will :)). This is where Text Wrangler enters the picture. You need to write your code using Text Wrangler, and execute it using Python. You do this python execution in the Terminal. To execute a python program, you need to run python from the directory (folder) that contains the program file. Open your Terminal and navigate to the appropriate directory (using cd). Then create and edit your file, saving it in the same directory.

[Demo this with a non-turtle program.]

Now we are ready to write our first program that uses the turtle.

```
# Stephanie Taylor
# first_turtle.py
# This is a simple program that demos the turtle.
from turtle import *

# Draw a wall
forward( 200 )
left( 200 )
forward( 200 )

# Wait for input
raw_input( "Press enter  " )
```

We need to import the turtle to had access to the function. Then we execute the drawing functions. Then we tell Python to wait for keyboard input, so that the program doesn't end and take down our drawing. The words after the pound sign are comments, which help human readers understand the purpose of the code.

For the next step in your project, you will be writing your own functions to draw shapes. Let's make an L shape.

You can define your own function like this:

```
# Draw an L with a 200-pixel long edge and
# 100-pixel short edge.
# Note: Draws the long edge first.
def drawL():
    forward( 200 )
    left( 90 )
    forward( 100 )
```

But there is something very limiting about my drawL function. Any time I call it, it will draw an L of the same size!!! I would like to draw an L that is resizable. To do that, I need to allow my function to take input. I will input the the distance of the short edge. I can define a parameter when I define

my function, by adding including its name in the parentheses. I then refer to the parameter by name in the function's code.

```
# Draw an L with the given short edge length.
# The long edge will be twice as long.
# Note: Draws the long edge first.
def drawL( short_edge_length ):
    forward( short_edge_length )
    forward( short_edge_length )
    left( 90 )
    forward( short_edge_length )
```

Here is an example of a file that contains both function definitions and the code to execute those functions:

```
# Stephanie Taylor
# l_drawing_example.py
# Draw a simple L using the turtle
from turtle import *

# Draw an L with the given short edge length.
# The long edge will be twice as long.
# Note: Draws the long edge first.
def drawL( short_edge_length ):
    forward( short_edge_length )
    forward( short_edge_length )
    left( 90 )
    forward( short_edge_length )

# main code
# Draw an L
right( 90 )
drawL( 100 )

# Wait for keyboard input.
raw_input( 'howdy! ')
```