1 Administrative Topics

- Did everyone make it to lab this week?
- Does everyone have a partner?
- My office hours must change today – they will be from 4-5 instead of 2-4.

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
- Arithmetic 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? I mentioned on Wed that precise thinking is important. 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
5 Python and its Memory Model

In this class, we will be learning the programming language named Python. I think of Python programs as a sequence of commands that manipulate data in memory.

To learn Python, we need to learn two things:

1. What are the legal Python commands?
2. How does Python view memory? I.e. what is Python’s memory model?

In the interpreter, I can show you a few Python commands:

• printing something to the screen: (print ' hello ')
• adding two numbers: (3+4)
• printing the result of a computation (print ”The sum is ”, 3+4)
• adding more numbers: (42+0.1)

That’s great, I can use Python like a calculator – I can type commands and it will do things. But what if I want to keep track of results from earlier computations? The answer is that I want to label the data. I want to create a named variable and associate some particular value with it. I do that with an assignment statement.

\[
a = 3 + 4
\]

This line of code is calculating the sum of 3 and 4, then placing that result into memory and labeling it as a.

We can retrieve the value from memory using the label

\[
\text{print a}
\]

Python programs generally consist of many statements that store and examine variables. And these variables are stored in tables. So, we can think of the basic memory model for Python as manipulation of tables.
In these tables, Python stores the names of values of all the variables. Values are things like numbers or strings. There are four “simple” types of data that Python manages:

1. Integers (<int>): whole numbers, e.g. -23, 0, 12324, 8
2. Floating Point Numbers (<float>): numbers with decimal points in them, e.g. 1.0, 5.3, 0.1
3. Booleans (<bool>): True or False
4. Strings (<str>): strings of characters, e.g. “hello”, or ’hi’

We will return to the concept of bool in a moment. First, let’s examine a table.

When we run Python in the interpreter, there is a “main” table (I often call this a “symbol table” because it keeps track of all of the symbols/labels). Commands manipulate the entries in this table. This table has two columns: Name and Value.

One example of this manipulation is by the assignment statement. An assignment statement associates data with a label, or variable. It does this by adding a row to the symbol table.

The following assignment statement assigns the value of 6 to a variable A.

\[ A = 6 \]

Note that this is not the same as “equals” in mathematics. Whenever you see this “equals” sign, you should say to yourself “this is an assignment statement”. We are assigning the value 1 to the variable A. I always say “gets”, e.g. “A gets 6”.

When this line is executed, Python checks the symbol table. If it contains an entry named A, it will update its value. If it does not contain and entry named A, it will add a row with the name A and value 6, e.g.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6 &lt;int&gt;</td>
</tr>
</tbody>
</table>

After executing the line
the table is updated:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5 &lt;int&gt;</td>
</tr>
</tbody>
</table>

### 5.1 Rules for Variable Names

A variable name can contain upper case letters (A through Z), lower case letters (a through z), decimal digits (0 through 9), and the underscore character.

A variable name must begin with a letter (upper or lower case).

Example variable names include, A, pic1, pic2, my_string, myString, etc.

Note that this means that the statement “6=A” is illegal in Python.

### 5.2 More about Types and Tables

I will illustrate the remaining points using an extended example. Suppose the symbol table begins empty.

Then we execute

```
georgeWeight = 26
```

to store the fact that my son George weighs 26 pounds.

The table becomes

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>georgeWeight</td>
<td>26 &lt;int&gt;</td>
</tr>
</tbody>
</table>

Now execute

```
georgeIsBig = georgeWeight > 30
```

Python compares the value of `georgeWeight` to 30, finds that 26 is not greater than 30. I.e. the statement that 26 > 30 is False, so result is the boolean value False. This value is assigned to the variable `georgeIsBig` and the table is updated:
Suppose George grows. We change his weight to reflect that growth:

\[ \text{georgeWeight} = 32 \]

and the value of `georgeWeight` changes in the table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>georgeWeight</td>
<td>32 &lt;int&gt;</td>
</tr>
<tr>
<td>georgeIsBig</td>
<td>False &lt;bool&gt;</td>
</tr>
</tbody>
</table>

Is George big? Well, yes, he is a big boy now. But the variable `georgeIsBig` is still False. The entry in the table is just data – there is nothing in the table that tells us how we got that value – their is nothing in the table that indicates one variable depends on another. It is up to us, as the intelligent humans to update it.

So we again execute

\[ \text{georgeIsBig} = \text{georgeWeight} > 30 \]

and the table becomes

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>georgeWeight</td>
<td>32 &lt;int&gt;</td>
</tr>
<tr>
<td>georgeIsBig</td>
<td>True &lt;bool&gt;</td>
</tr>
</tbody>
</table>