

PUTTING SCIENCE ON THE WEB

Kevin Cooper

Washington State University

Table of Contents

1 The Internet.....	3
1.1 Packets.....	4
1.2 The Network.....	6
Internet Terms.....	6
2 The Browser.....	7
2.1 Browser Basics.....	8
2.2 Browser Motivation.....	10
2.3 Using a Browser.....	16
Nomenclature.....	16
2.4 Editing Web Pages Using a Browser.....	16
Netscape Composer Commands.....	20
2.5 Hypertext Markup Language.....	23
HTML Tags.....	31
2.6 Cascading Style Sheets.....	33
Style Sheet Properties.....	35
3 Mathematics.....	38
4 Graphics.....	39
5 Interaction.....	40

1 THE INTERNET

Twenty years ago, a computer network was composed of a single large machine connected by long sets of wires to a number of small green-screen terminals. The terminals were only capable of displaying twenty-three lines of text using a restricted character set. The most exciting form of communication that took place over the computer network was the Unix utility called "talk", which permitted two users of the same computer to type text messages to each other.

Clearly, times have changed. No one can work in science or engineering today without a computer nearby. Electronic mail, typesetting, and web services accelerate the flow of information to speeds unimaginable when we were born. A variety of data compression schemes, coupled with large-bandwidth networks, make transferring images as easy as transferring text. The most humble personal computer today has more power and memory than that mainframe that supported 100 green-screen terminals in 1980.

What is the Internet? The name itself says much about it: it is a network that connects networks. In other words, while one may have a local network anywhere - in the office, in the home - that network is a world unto itself unless it is connected to other networks. The Internet is the infrastructure that permits you to connect from your local network to machines all over the world. The infrastructure is composed of cables, network devices that switch and route the signals to the next station, and the collection of languages and programs that provide the languages for these devices to work in.

There are many kinds of media used to create computer networks, from coaxial cables made of copper to fiber optic strands. To some extent, the properties of the media determine the types of packets that are sent across them, but different packet technologies may use similar media. For example, fiber optic cables are used for FDDI packets on some networks, for ATM transmissions on others, and for fast ethernet packets on others still. The key to the Internet is that all of these different media, transmitting different kinds of packets, may still communicate through the use of standardized protocols, such as TCP/IP, and at higher levels, FTP and telnet.

1.1 Background

Why is it that so many things in computing seem to be organized into fours, eights, and sixteens? What is the difference between ethernet and Internet? In spite of efforts to make networks and the Internet transparent to users, there remain many places where a little bit of understanding of electronics and binary arithmetic can help you make some sense of what is happening.

At the bottom of the network you use is the *physical layer*. This is the actual cable, network interface cards, and other hardware that is required to make the network function. There are many different styles of cable used on computer networks, and different kinds of communication used on those cables, but many of them rely on sending low-voltage electrical signals over the cables. Those voltages are always interpreted in a binary sense - either on or off. If a voltage is "on", then it is interpreted as a one, while if the voltage is "off", then it is interpreted as a zero. Of course, there are some network media that use light instead of electrical signals to record those "on" and "off" notions, but logically, the idea is the same.

Since every signal is composed fundamentally of ones and zeros, then mathematically we clearly are working in base two. For anyone familiar with arabic numerals, this should present no problem, but let's review it here. When we write a base two numeral, the only digits permissible are ones and zeros. As in a decimal representation of numbers, the position of the digits has a meaning in terms of the power of the base that the digit is multiplied by. More simply, in this case the rightmost digit multiplies 2^0 , the second digit from the right multiplies 2^1 , the third from the right multiplies 2^2 , and so on. Here are some examples.

$$100 \text{ (base 2)} = 1x2^2+0x2^1+0x2^0 = 4 \text{ (base 10);}$$

$$101 \text{ (base 2)} = 1x2^2+0x2^1+1x2^0 = 5 \text{ (base 10);}$$

$$111 \text{ (base 2)} = 1x2^2+1x2^1+1x2^0 = 7 \text{ (base 10);}$$

$$1001 \text{ (base 2)} = 1x2^3+0x2^2+0x2^1+1x2^0 = 9 \text{ (base 10);}$$

$$10011001 \text{ (base 2)} = 1x2^7+0x2^6+0x2^5+1x2^4+1x2^3+0x2^2+0x2^1+1x2^0 = 169 \text{ (base 10).}$$

In computing, each digit of the binary representation is called a *bit*. These bits are typically organized further into groups of eight (2^3), called either *bytes*, or in the context of networking, *octets*. Since eight binary digits are sufficient to represent numbers from 0 to 255, inclusive, such numbers appear prominently in all aspects of computation. There are several ways to write these numbers. Obviously we can use a decimal representation as we have above, but it is equally common in computing to use a *hexadecimal* representation.

Hexadecimal means: "base sixteen". Writing hexadecimal numbers is somewhat problematic - we need digits to represent eleven through fifteen. It is traditional to use the letters a through f for this purpose. In other words, if we were to count in hexadecimal, it would go like this: 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, 10, 11,...

This all fits together nicely with base two: $16 = 2^4$. Thus, any hexadecimal number may be represented as a four digit binary number (four bits), and conversely, four bits may be represented by a single hexadecimal number. Moreover, $256 = 16^2$, so that we may use one hexadecimal number to represent the upper four bits of an octet, and a second hexadecimal number to represent the lower four bits. The position of the digit again signifies the the power of sixteen by which it is to be multiplied. Here are more examples:

$$10000000 \text{ (base 2)} = 128 \text{ (base 10)} = 80 \text{ (base sixteen);}$$

$$10011001 \text{ (base 2)} = 169 \text{ (base 10)} = 99 \text{ (base sixteen);}$$

$$11110001 \text{ (base 2)} = 241 \text{ (base 10)} = f1 \text{ (base sixteen);}$$

$$11010 \text{ (base 2)} = 26 \text{ (base 10)} = 1a \text{ (base sixteen).}$$

These hexadecimal numbers appear often in computing, precisely because octets are used so often. For example, hexadecimal numbers are typically used in writing down addresses for ethernet networks.

Ethernet

The most common type of local network is called *ethernet*. This word actually describes a collection of different media and signal types. All these different technologies are united in the sense that they all depend on *packets* of a certain form. A packet is a sequence of ones and zeros that have a particular significance - if you will, an electrical or optical signal with a beginning, some content, and an end. Each packet contains an address for the local machine that is its immediate destination. That address is called the ethernet address, or more often, the MAC (Media Access) address. The MAC address is a 48 bit binary number. In writing the number, we don't want to use the space and time required to put down each of the ones and zeros, so it is organized into six octets. Typically, each octet is written using hexadecimal notation, and the octets are separated by colons. Thus, MAC addresses are typically written in forms such as 08:00:a0:b1:18:03 or 00:05:15:64:68:f0. We see that the 48 bit address is written as twelve hexadecimal digits.

It is worth noting that MAC addresses are purchased in blocks of 16,777,216 at a time by the manufacturers of ethernet interfaces. This means that the first three octets in the MAC addresses identify the manufacturer of an ethernet card, while the last three octets are simply a number from within that sequence, that is specific to the card.

Every ethernet packet contains either a MAC address of its immediate destination, or an address that indicates that it is intended to be received by every computer on the local network. In this way, the receiving computer understands which of the packets on the network are intended for it, and which are not. Each ethernet packet also contains the address of the computer that sent it, so that if a reply is needed, it may be sent.

There are many other types of computer networks, but we do not need to consider them or the details of ethernet networks. Our point here was simply to show the way that the physical network of cables and appliances fits together with the logical network of ones and zeros.

Colors

Another place where hexadecimal numbers show up often is in the specification of colors on the Internet. Most likely you have seen a monitor that claims to display millions of colors, or perhaps claims exactly 16,777,216 colors. It is not a coincidence that this number is exactly 256^3 . Colors are specified on computer screens as combinations of red, green, and blue, or in computer-speak: RGB. The intensity of each color component is typically an eight-bit number - an octet. Again, this is frequently written using hexadecimal notation. Thus, it is common to see a color written as a six "digit" hexadecimal number. The first two digits denote the intensity of the red component of the color, the second two denote the green component, and the third pair denote the intensity of the blue. Some examples of color specifications follow:

ffffff - white;

000000 - black;

ff0000 - red;

600000 - dark red;

006000 - dark green;

ffff00 - yellow;

e0e0ff - light blue.

The last item from the examples points out a peculiarity of color specifications: the smallest number from among the red, green, and blue components determines the amount of white (or gray, if you will) in the color. In particular, the last color should be viewed as `e0e0e0+00001f`, or in words, as light gray with a little bit of blue.

1.2 Packets and the TCP/IP Suite

In July of 1961, a computer scientist at the Massachusetts Institute of Technology published a communication discussing the idea of "packet switching"¹. This was a landmark, inasmuch as this idea forms the basis for most computer networks since then.

There are two basic forms of communication between computers. *Connection-oriented* networks work by forming fixed connections between the computers, dedicated to the conversation between those computers. During the time that the connection exists, the path between the computers is unchangeable and guaranteed. No other computer can interfere with the interaction. On the other hand, it is expensive to maintain enough lines to carry the connections required, and once the lines available are all in use, then no further connections may be established. The telephone network is an example of a connection-oriented network. To see the chief disadvantage of connection-oriented networks, just think of busy signals.

In contrast, a packet-switched network is connectionless. The basic idea behind packet switching is to send information in small batches, called (of course) packets. A message may be broken into these small pieces and wrapped in information about its source and destination. Each packet might travel by separate routes to its destination, where information it contains may be used to reassemble it with other packets to form the entire message. Each packet can be formed in such a way as to show whether it remained error-free, and indeed can be put together to allow a certain level of error correction. If a packet was damaged or lost, it can be sent anew, without having to send the entire message.

The advantage of this approach is clear enough: a single line may be shared among many computers. The disadvantage is equally clear: if too many computers try to use the line, then all communication will be slower - indeed, all traffic will be slowed. To imagine this, one might think of a party. If there are not too many people at the party, then small groups may form to carry on conversations, each party ignoring what it hears from the other. However, if the party is larger, then the conversationalists have increasing difficulty hearing each other, and frequently speakers must repeat themselves. The actual network has a worse time with this, inasmuch as the capacity for carrying the packets has an absolute limit (unlike the party).

These concepts were developed through the decade of the 1960s in three separate laboratories, independently. Indeed, the researchers involved were unaware of each others' work until 1967. In 1967, a paper was presented providing a view of what a packet-switched network might look like, and the ARPANET² was born.

ARPA, or the Advanced Research Projects Agency, was the governmental body responsible for a number of advances in computer science in the 1960s and 1970s. The ARPANET was its creation, perhaps its most notable achievement. By the end of 1969, four computers were able to communicate with one another on this precursor of the Internet.

If a message is to be broken into packets, obviously there must be some uniform way of doing this so that the receiving machine can reassemble the message from the packets. The initial ARPANET used rules encapsulated in NCP - Network Control Protocol - to perform the tasks of disassembly and reassembly. However it became clear early that this protocol was limited, and that it would be necessary to extend it or to create a new protocol governing this aspect of the network. For this reason, in 1972 Bob Kahn formulated principles for a new set of packet-

¹ L. Kleinrock, "Information Flow in Large Communication Nets", RLE Quarterly Progress Report, July 1961.

² L. Roberts, "Multiple Computer Networks and Intercomputer Communication", ACM Gatlinburg Conf., October 1967.

handling protocols, known now as Transmission Control Protocol and Internet Protocol - TCP/IP³. This combination is now the basis for much of the traffic on the Internet.

Obviously, there are two separate protocols involved here. IP (Internet Protocol) is the part that deals with "long range" addressing, while TCP (Transmission Control Protocol) manages the connectionless session. In other words, IP is used to provide each packet with the information needed to get it to its final destination, while TCP is at a higher level, breaking the message into packets, keeping track of whether all the packets have been delivered, requesting that packets be sent anew, if necessary, and reassembling them at the destination.

We don't need to know too many details of this process, but in order to clarify this somewhat, consider an email message from a machine called `mymachine.edu` destined for `yourmachine.com`. The email program assembles the message and passes it, together with the address of the destination, down to the TCP protocol. TCP breaks the message into packets, and places the address of the destination computer into each packet. It also places information into each packet regarding where that packet fits into the whole. After that, it puts each packet onto the network. Each packet moves along the network to various nodes, called routers, that determine where it should next be sent. As a result, the packets may take different routes to `yourmachine.com`. For example, the first packet may go to `yourmachine.com` via a router called `georgia.edu`, while the second packet sent may go via `carolina.mil`. The third packet may go via a route that is currently not working, and disappear. The result of this is that the second packet sent is the first to arrive at `yourmachine.com`, followed closely by the first one sent. Each time a packet arrives, the TCP software on `yourmachine.com` sends an acknowledgment packet back to `mymachine.edu`. When `mymachine.edu` has waited some period of time without receiving the acknowledgment of packet number three, it sends that packet again. Eventually `yourmachine.com` recognizes that all of the packets have arrived, assembles them back into the entire message, and passes that to its email handling program for delivery.

There are a couple of external things required for all of this to work. There must be a coherent addressing system for the Internet; there must be a computer somewhere that understands how to translate names of computers into those addresses. Finally, each of the computers at either end of the transaction must have software to handle the TCP protocol, called a TCP stack.

The addresses used are called (surprise!) IP addresses. IP addresses are very simple - they comprise four octets. In another way of looking at it, an IP address is an ordered collection of four numbers between zero and 255, inclusive. For reasons that are mired in the history of the Internet, when IP addresses are given it is traditional to represent each octet of the address in decimal form, instead of its hexadecimal form. Thus, typical IP addresses look like `191.16.1.1`, or `134.121.43.233`.

Humans generally have more trouble remembering sets of numbers such as those in IP addresses than they do with names. For that reason, we have all grown used to the idea of giving the names for the computers we want to contact: for example, www.amazon.com, or www.sci.wsu.edu. Those names do not go into the packets sent to those computers. Instead, the program sending the packets first asks a *nameserver* to find the IP address associated with the name given. The nameserver is simply a computer that is in charge of the names of certain groups of machines. No single computer knows the names of all the computers on the Internet; they simply know which other computers to contact in order to get those names and addresses.

Ethernet or other packets with IP information alone would have all they need to get to their destinations. Unfortunately, networks are a dangerous place: variable electric fields or poor connections can introduce errors in packets; two or more packets can try to share the same network segment, leading to a *collision*; routers and other network devices can malfunction, causing packets to be lost. If packets only carried addressing information, only a fraction of them

3 R. Kahn, Communications Principles for Operating Systems. Internal BBN memorandum, Jan. 1972.

would get to their ultimate destinations. The TCP protocol was developed to provide reliable transport of collections of packets over an unreliable network.

We are not interested in the details of the TCP protocol here, so we only provide a very brief summary of the ideas. Simply put, TCP breaks the data into bits, encapsulates those bits into packets, together with information regarding their position within the whole and the IP address, and then sends off the packets to their destination. Along the way, some of the packets are lost and some arrive out of sequence. Nonetheless, TCP on the receiving computer sends an acknowledgment packet for every packet received. Some of those packets are lost in turn, but as those acknowledgments that do arrive come in, TCP on the sending computer checks off the packets it knows arrived. After a timeout period, it sends those for which it has not received acknowledgments anew. Since some of the original packets did arrive at the destination, and it was the acknowledgments that went awry, the result is that some packets are received twice or more. TCP on the receiving machine simply discards the excess packets, sends acknowledgments again, and begins to reassemble the content.

Thus, the job of the sending computer is to keep sending packets until all are acknowledged, while the receiving computer must send acknowledgments for each packet that comes in. This simple-sounding combination of tasks is actually quite complicated to maintain in practice, and thus TCP is not a trivial protocol by any means. This also means that the network is filled with packets that do not contain content of the information being transferred. Finally, it is a good deal of work for both machines to keep track of all the incoming and outgoing packets.

On the other hand, not every network application requires the reliable session that TCP provides. For such applications, other less reliable protocols suffice. For example, the User Datagram Protocol (UDP) is used for many file services and similar applications in which it is not necessary to have a "session". For example, in Unix, the network file service uses UDP to maintain contact with a server. In that situation, if files are not actually being transferred then all that is required is for the client machine to keep track of the server machine, to be sure that it remains up and is willing to serve the file system in question. UDP does not use acknowledgment packets to maintain the session, and it does not keep track of the order of incoming packets. The advantage is speed. However, UDP still uses IP to manage the addressing, and so it is considered to be part of the TCP/IP suite of protocols.

Internet Terms

Bit - A single binary digit.

DNS - Domain Name Service. This is the protocol that governs requests and answers associating Internet names with IP addresses.

Ethernet - A specific type of network arrangement used by most universities and businesses within their facilities. It is packet-switched, and can be sent over a variety of media, at various speeds that depend on the media.

IP address - A four-octet number that identifies a unique computing system on the Internet. IP stands for Internet Protocol.

Nameserver - A computer that runs a program that knows how to associate an IP address with an Internet name. Nameservers operate based on the DNS protocol, and indeed, are sometimes known as DNS servers.

Octet - Packets are generally formed from groups of eight digits - ones or zeros. These groups of eight are typically called bytes in the computer world, but in computer networks, for reasons that are unclear, they are usually called octets instead.

Protocol - A language that computer programs use to communicate with each other. The word refers specifically to languages used over a network.

RFC - Request For Comments. These form the alarmingly democratic structure governing standards and documentation for the Internet.

Port - Some network transport protocols associate certain programs with numbers. Even though the network hardware receives all of the packets through the same connection, it sorts the packets into logical mail slots for their destination programs according to these numbers. These logical mail slots are called ports.

1.3 The telnet and ssh protocols

TCP/IP provides the means of getting your information from one computer to another, but what is that information? TCP/IP carries the information from many different, higher-level programs. The first of these to be standardized was the telnet protocol. Telnet very simply allows you to execute commands on a remote computer. This means more and less than a generation raised on the Microsoft Windows interface might understand. When telnet first came into use, all computers interacted with a user purely through a prompt at which the user would type text commands. For example, to list her files, the user might type the Unix command `ls -l`. The telnet protocol did no more than codify a means of allowing the user to execute such commands on a machine far away.

To use telnet, you do not need to do any more than start a shell on your local computer, and then type `telnet remotecomputer`, where `remotecomputer` is the Internet name or IP address of the computer you want to log into. If the computer accepts the connection, then it should invite you to enter your user name and password, and it should give you a command prompt. You are free to run any commands on that remote computer that you would be able to if you were sitting at its console.

We should note here that there are actually two things happening in a telnet session. First, you are running a program on your local computer which in turn is talking to a program on the remote computer. The program on your computer is called a Telnet client. In general, when we use the word client we mean a program on your computer. The program on the remote computer that responds to your Telnet request could be called a Telnet server, but it is usually referred to using Unix parlance, and is called a telnet daemon. The two programs communicate using the Telnet protocol. It is the Telnet protocol which is standardized through the RFC process. Thus, while there are many different telnet clients which may incorporate more or fewer features, there is effectively only one Telnet protocol. All of those Telnet clients should use the same protocol.

Some computers are not good at starting shells where you can type "telnet". Instead, such computers frequently allow you to start a telnet client in a new window. In this case, you click a menu item or double-click an icon labeled "Telnet" or "Terminal" or something related. In particular, Windows ships with a default telnet client that runs in its own window on the screen. You may start it in any of several ways, but it is probably easiest to click [Start->Run] and type `telnet remotecomputer` in the text box of the dialog that results.

Sometimes you want to log into the remote computer using an IDE different from the one on your own computer. Telnet provides for this by including a switch to allow you to change IDs. The syntax is `telnet remotecomputer -l userid`, where user ID is the name of the user that you want to use of the remote computer. In other words, this command will log you into the computer `remotecomputer` using the name `userid`.

There are three points worth making here. First, it is probably important to distinguish between the telnet client and the telnet protocol. The client is an actual program that runs on your local machine that knows how to use the telnet protocol to communicate with remote computers. The protocol is the high-level language that the client uses. The protocol is standardized in RFCs _____, but there are as many different clients as there are dandelions in your yard. The second point is that the ability to run a telnet client does not correlate with the ability to run a telnet server. For example, Windows computers generally do not allow users to run shells remotely. Even Windows NT servers only allow users to use shared file systems; they do not ordinarily allow users to run command shells. This changes in Windows 2000.

This leads to the third point. Telnet grew out of the Unix world, and developed very early in the Internet world. That is why it is command-line-oriented, but it also explains the comparative lack of security it embodies. If your computer provides telnet services, that means that it has a TCP port open to the world. That telnet port may be used by anyone to try to log into your computer.

The only security you have lies in requiring users to give a password to log in to your computer. Unfortunately, even that security is minimal. Anyone who has access to any computer on your local network can run a program called a "sniffer" that examines all the TCP packets that go by. When it finds the one containing your password, it logs it. The owner of the sniffer comes back later to examine the log (or has it mailed to him), and then he has your password. This is only the most obvious way to compromise a telnet service. There are many others.

To address this glaring insecurity in telnet, Secure Shell protocols have been developed. These are called collectively "ssh", and the only thing they add to the telnet service is encryption. If you use ssh, then instead of sending your password and other information across the network in plain text, you send an encrypted version. Presumably the person running the sniffer cannot decrypt the password in a short enough time for it to be useful, so you are more secure.

Ssh works using "public-key encryption". In other words, the key to the code is public. This may require some explanation. Public key encryption actually relies on two keys: what is public, while the other is private. You are free to send the public key to anyone and they can use its to encrypt data to send back to you. On the other hand, once encrypted the data can only be unencrypted using your private key. Thus, in the case of SSH, you send your public key to any machine with which you want to communicate, and thereafter the session is encrypted using that public key. Only your computer can understand the communication after it is encrypted, because only your computer knows the private key.

Using SSH is almost completely analogous to using Telnet. In particular, the basic format to use SSH is `ssh remotemachine`. Again, if you want to login using a user ID that is different from the one on your own machine, then you may use the `-l` option.

There are two versions of SSH. SSH 1 uses 56 bit encryption. That means that the public key is a number that can be expressed using 56 binary digits. SSH 2 is a newer protocol that uses 128 bit encryption. SSH 1 has become common on many machines around the world, but SSH 2 is less common and usually more expensive. Windows clients for SSH 2 are usually not free. When you need to distinguish between SSH 1 and SSH 2 it is always safe to call it as `ssh1 remotemachine`.

1.4 File transfer protocol

Telnet allows you to run commands on a remote machine. Unfortunately, that doesn't permit you to run the commands and then send results back to your home computer for analysis. This is a kind of task that occurs very often in scientific applications; you want to use some powerful machine at another university, but you want to produce the graphics and reports on your desktop machine or on your home computer. To facilitate the transfer of files from one computer into another, file transfer protocol (FTP) was created.

File transfer protocol actually allows you to run many commands on the remote computer. For example, you can list the contents of a directory, you can change directories, and sometimes you can even move files from place to place on that computer. On the other hand, it does not give you complete control on the remote machine. It is designed to allow you to manipulate files and to transfer them to your computer but not to allow you to run programs or perform other tasks on the remote computer.

FTP first appeared in 1971 in RFC 114. That means that ftp antedates TCP. Indeed, the first versions of FTP used data transfer protocol to maintain the connections required to send files across networks. Version 2 of FTP appeared in 1971 also, followed by version 3 in 1972. The version of FTP we use today is numbered 5.1, and the standards for that date from 1985.

FTP is similar to telnet in that there is an FTP client running on your computer, and an FTP daemon running on the remote computer. The FTP daemon communicates with your computer using the FTP protocol. In short, we must again be careful to distinguish between the protocol and client programs.

There are many sophisticated client programs that use the FTP protocol. However, at bottom they all use a relatively small command set that is shared with more primitive FTP clients. They all do the same jobs. They all have roughly the same capabilities. We will begin by discussing the use of a primitive FTP client, and later finish this section by discussing one example of a more sophisticated client. We will not discuss details of the protocol itself.

Basic FTP commands

Because FTP is so old, many FTP clients still use a command-line interface. Such interfaces are very primitive, but are still effective today. In this section, we discuss some of the basic commands found in any FTP client. In this discussion we will suppose that we are looking for some free software that plots graphs of functions. We will look for the software on the computer `freesoftware.org`. That computer is imaginary. Then again, it probably isn't. Since many of the earliest FTP clients ran on Unix computers, the command-line interface to those clients is very like that of Unix. In particular, commands have the form `commandname option1 option2 ...`. The options are separated by one or more spaces.

Obviously, the first thing we have to do with FTP is to make a connection to a remote computer. We initiate this process in any of several ways. For example, for almost a command-line interface we type `ftp freesoftware.org`. This establishes an FTP connection with the machine `freesoftware.org`, which asks us for login information. We enter our user ID and password to establish the connection. Alternatively, we could simply type `ftp` without any mention of the name of the remote computer. At that point we get an FTP prompt, at which we enter the command `o freesoftware.org`. The command `o` stands for open, meaning that we wish to open a connection to remote machine. Once again the remote computer will prompt you for your user name and password to complete the connection.

We should note that there are many so-called anonymous FTP sites on the Internet. They are called anonymous because you do not need to have an account on those computers in order to make FTP connections. Indeed, anyone can log into those computers and retrieve files from them. Such sites are usually used to distribute free software or documentation about software.

Usually they invite you to enter anonymous as login ID, and to use your email address as your password, so that they have some record of who's using the site. We will suppose that freesoftware.org is such a computer, so that after we opened the connection to that computer, we typed "anonymous" for our user ID, and yourname@yourmachine.edu for the password.

Now that we are connected to the computer, we need to see what directories are available from which we can retrieve files. We can always list the contents of a directory by using the ls command. Ls simply stands for "list". Of course, FTP is designed to transfer files, so that in typing ls, we are actually asking the FTP server to create a file composed of the contents of the current directory and send it to us. On the other hand we are asking the local FTP client to take that file and display it on the screen. These details explain the peculiar messages that appear on the screen in the course of the FTP transaction. In the case of our example, the screen that results from our ls command looks like the following.

```
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls (192.168.1.27,4797).
d--x--x--x  2 0          0          8192  Aug 27  1999 bin
dr-xr-xr-x 25 0          0          8192  Nov 16  08:31 pub
226 Transfer complete.
```

The first-line says the command was successful, the second indicates that a file is being sent from freesoftware.org to our machine. The file contains the result of the command `/bin/ls`. At the end of that line we find the IP address of our machine, and the local TCP port number used for the connection. We see that that the `pub` directory looks promising, so we can change to that directory by using the `cd` command. `Cd` stands for change directory. We type `cd pub`. After that we use the `ls` command again to list the contents of that directory, and continue in that way until we find what we want.

When we find the directory that contains the file we want, then we can retrieve that file using the `get` command. The `get` command brings the file to our computer and puts it on the current directory in our local computer. You might ask, "what is the current directory?" Since we're assuming that we are running the FTP client program from a command line, then the current directory is simply the directory we were in when we started `ftp`. Thus, supposing that the file we want is called `libraries.doc`, then to retrieve the file from the directory at freesoftware.org that we are looking at, and to put it in the current directory on the local computer, we need only type `get gnuplot.tgz`.

It frequently happens that we start the FTP client without thinking about what directory we are in. We find the file we want to retrieve, but then realize that it will go to the wrong directory, or even to a directory that we are not allowed to write to. We need to change the local directory to deal with this problem. To do that, we use the `lcd` command. `Lcd` stands for "local change directory". In our example, we had been looking for plotting programs on our Unix machine, in the `/usr/local/bin` directory. We found that we lacked any decent programs for our task, so we logged into the anonymous FTP server to get that library. Unfortunately, now our local current directory is `/usr/local/bin`, on which we do not have permission to write. To change it, we just type the command `lcd /home/mydirectory` to get to our own directory, and then type `get gnuplot`.

The `get` command can take two arguments, if you want to specify the local file on which to place what gets transferred. For example, to get the `gnuplot` file above without changing directories all the time, you might use the command

```
get /usr/local/bin/gnuplot /home/mydirectory/gnuplot.mine.
```

This gets the file from the remote directory, and puts it directly on your home directory with the name "gnuplot.mine".

Unfortunately, we probably find after we get the file that we cannot use it. The reason is that the default method of transfer for primitive FTP clients is to send all files as if they were composed of plain text. We don't need to know the details of what that means, aside from noting that when the file is compressed, or is a program, or is anything other than plain text, then the transfer is incomprehensible to the receiving computer. To deal with this, we must tell this client that the file to be transferred is not a text file. We do that by using the command `binary`. In other words, before the `get` command, we type `binary`.

Sometimes we want to get several files off of the remote computer. If they have similar names, we can use the `mget` command together with wildcard characters to get the collection. In our current example, we realize when we go back to `freeware.com` that in addition to using plain text mode to transfer the file we wanted, we also neglected to get a collection of interface functions that `gnuplot` needs. Therefore, after typing `binary`, we get the files using the command `mget gnuplot*`. This command brings us both the `gnuplot` file and the `gnuplot_x11` file. As FTP gets each file, it asks us to verify that we want it.

The asterisk in the above command is called a *wildcard*. It signifies that `gnuplot*` matches any string that starts with the word "gnuplot". Thus it matches the names `gnuplot`, `gnuplot_x11`, `gnuplot_my_foot`, and `gnuplot_is_a.very.fine.plotting.program`. It does not matter how many characters we put in place of the asterisk - it matches anything.

When we are finished transferring files, we exit the FTP client using the command `quit`. This simply terminates the connection. It does not save anything, or ask you for verification.

FTP commands

binary - this command tells FTP to interpret the file in a raw form. The default is to treat the file as plain text. We use this command when we want to send executable files or compressed files. It takes no arguments.

cd - To change directories on the remote machine, use this command. To change directories relative to the current directory on the remote computer, use the form e.g. `cd mydirectory`. To make a change to an absolute path, preface the directory specification by a slash: e.g. `cd /pub/subdirectory/mydirectory`.

get - This is the basic command to retrieve a file from the remote computer. The syntax is `get remote_file local_file`. You may leave off the second file name, if you want to store the file on the current local directory using the same name.

hash - This peculiar little command causes hash marks to be printed at intervals during the transfer of a file. The marks give you an idea of how fast the transfer is taking place, and give you assurance that the operation is proceeding, or let you know when it has stagnated.

lcd - To change directories on the local machine, use the `lcd` (local change directory) command. Again, use either relative or absolute specifications for the path to the directory.

ls - This command lists the contents of the current directory on the remote computer. There is no corresponding command for the local machine.

mget - You can transfer multiple files from the remote machine to your local computer using this command. Typically, one would use this in conjunction with wildcard characters. For example, `mget /pub/mydirectory/myfile*` would get all files whose names start with the letters "myfile".

mput - Similarly to `mget`, the `mput` command allows you to transfer several files from

| your local computer to the remote machine.
put

|

2 THE BROWSER

When Tim Berners-Lee developed the first browser program in the early 1990s, it changed the Internet forever. Before the browser, our experience of the Internet consisted entirely of interaction with primitive command-line tools. We had to memorize a number of arcane and limited commands in order to interact with various separate programs that performed different functions. The browser brought a number of different Internet protocols into a single program, and made our interaction with that program much simpler. The result was that command shells, file transfer, and text and graphics display could all take place in one program, giving a life to the network that had not previously been there.

2.1 Browser Basics

What is this thing that made a computer network come alive? It is nothing but a program that is capable of processing and displaying a large variety of data. There have always been many different kinds of files available on computers. Examples are given in Table 1. The problem for users was that it was difficult to determine which files were programs and which were input to programs, or output from programs. Moreover, different file formats required different methods of transfer from one computer to another. Data files might be used on many different operating systems, while computer programs were highly dependent on operating systems and hardware architecture. In order to view formatted text or an image file, one needed to install the viewing program on one's own computer.

1 Table

File Type	Description
Flat Text	These files are simply character data that a human can read. Examples might include simple "README" files, data files for programs, or even TeX files or HTML files.
Formatted Text	These are files containing special characters that give format instructions to a display/printing program. For example, Unix nroff files, Microsoft Word files, and other such documents include a good deal of text, but also contain non-printing characters.
Binary Files	These are, for the most part, computer programs containing machine executable instructions.
Images	These are a kind of binary files used as input to image display and manipulation programs.

The problems in networked computing were exacerbated by the plethora of protocols required to transfer data from one computer to another. Some of these protocols are listed in Table 1.2. The different protocols had different functions, and the programs that implemented them used different syntax. This made the problems of networked computing comparable to those experienced by a world traveler. For example, suppose that you are a researcher in physics at UCLA in 1988. You want to run some programs on a computer at Argonne National Laboratories, and then analyze the data on computers at your university. You must first log on to the local computer, which runs an operating system called VMS, which you must use fluently. Then, you must start a program that supports the telnet protocol to log in to the machine at Argonne. Next, you use the command line for the operating system at the Argonne machine (Unix) to run your program, producing a data file. Now you must transfer the file back to UCLA, so you log out of the telnet application, and start one that supports FTP. You use a different syntax in this program to retrieve the file you created at Argonne, and you are finally ready to analyze the data using a plotting package on your machine. To summarize, to accomplish this transaction, you must know the syntax of VMS, Unix, a telnet application, an FTP application, and your local data analysis/graphics program.

2 Table

Protocol	Description
Telnet	Allows a user to run a shell on a remote computer.

<i>Protocol</i>	<i>Description</i>
FTP	File transfer protocol - used to transfer files of all types from one computer to another
HTTP	Hypertext Transfer Protocol - also used to transfer files from one machine to another, but they files must be in a special format.
SMTP	Simple Mail Transfer Protocol - the language of electronic mail.
Gopher	

Browser programs did not change these facts, but they were able to listen over the network and speak the language of the machines they found. They could understand when to format output, when to display an image, and when to save a binary file. This ability to deal with multiple protocols, and to interpret many different file formats opened up the internet to such vastly expanded possibilities that it took on a new name: the World-Wide Web. In modern times, of course, "surfing the Web" is an activity pursued by everyone, from schoolchildren to participants in elder hostels, from American students to Malaysian businessmen. This access is the direct result of the invention of the browser and the attendant server software and infrastructure.

2.2 Browser Motivation

There are really two parts to the act of surfing the Web: it requires two computers running programs that can speak to one another. The first is the server. The server is a computer or program that listens constantly for requests for files, and sends those files to the requesting program. The part that you interact with is, of course, the browser. The browser is, in a sense, the boss of the transaction. The browser makes the request, and the server is but a secretary who fetches and sends the information. The browser has millions of such secretaries at its disposal. On the other hand, the computer where the browser runs does not need to be as sophisticated as the server computer, since the server must listen for and handle a wide variety of requests.

You have probably already noticed that we use the word server in two different ways in this discussion. On the one hand, the server is an entire computer, complete with an operating system that is willing to offer information through various TCP ports that are associated with various protocols. On the other hand, we frequently use the word server to mean specifically the program that listens for and responds to hypertext transfer protocol. Thus, it is possible for your browser to contact the server computer without ever talking to the HTTP server. In order to attempt to avoid confusion, we will try always to refer to *the server computer*. When we use the term "server" without qualification, we mean the HTTP server program.

URLs

The heart of the transaction between the server computer and browser is the Universal Resource Locator (URL). This is the information that the browser uses to find the required file on the appropriate directory on the server, and to transfer that information via the appropriate protocol. The URL has four basic parts: a protocol specification; a machine identifier, possibly combined with a port specification; a path component to indicate the location of the file on the server machine; and finally, either bookmarks within the file or input to the program that runs on the server. This is diagrammed below.

```
protocol://server.name:port/path/to/file#bookmark  
protocol://4byte.server.ip.address/path/to/file?input
```

You have probably typed URLs into your browser manually before. It is easy to do using a text box on the toolbar of most browsers.

The protocol tells the browser, in essence, what language to use in contacting the server computer. This also carries implications for the TCP port that is used. Typically, the server computer runs a number of different programs, one of which will handle FTP requests, with perhaps another for gopher requests, another for telnet or SSH, and another for HTTP. The browser program is able to negotiate with all of those different server programs to get the information it needs.

The most common protocol used by browsers is HTTP: Hypertext Transfer Protocol. This program allows files to be transferred, but also negotiates the format in which those files are transferred, and provides information about what to do with the files after they are delivered. You will also see the word "file" appear in the protocol specification frequently. This does not actually denote a protocol, but instead indicates that the browser should open the file on the local computer directly, without using the network at all. Note also that when you type the URL into the browser directly, you do not usually need to type http if that is the protocol you want to use - it is usually the default protocol for the browser.

After the protocol specification you must identify the machine you want to contact (unless you used file request instead of a protocol). You may identify the computer using either its Internet name or using its IP address. Typically the machines you want are named www.sitename.com, or something similar. Most often, the www is not the real name of the machine - instead, the

machine might have various aliases. To make matters more confusing, in modern times there are many "web hosting services"; companies that sell an address associated with a block of storage on one of their machines. As a result, such a computer might have many (thousands) of aliases that do not even share a site name. Indeed, the aliases may be associated with individual file systems on the computers. In other words, while we have called this part of the URL the machine specification, it is actually more complicated than that. On the other hand, we do not have to worry about that - we may as well treat this as the name of the computer. Most of the time, it is.

The next part of the URL does not always appear. By default, HTTP is associated with port 80 on the server computer. However, it is possible to set up a server program to listen to almost any port you desire. Ports near 8000 are commonly used. If the port has a number other than 80, you must specify that number after a colon following the computer name.

Following all of the computer information comes the part of the URL where we tell the location of the file we are trying to reach on the computer. We do this by listing the file name, its directory, and all of the parent directories of that directory, starting from the top-level directory. There are a number of things to note about this. First, the top-level directory may not be the root directory for the computer. The administrator of the server computer may designate any directory she likes as the *document root*. A common choice on Unix computers is to use the /pub directory as the document root. Only subdirectories of this directory can be viewed using a browser. This provides the first small level of security for the server computer: not all of its files are visible to those using browsers.

Subdirectories of the document root are separated by forward slashes (/), regardless of the syntax in which that is done on the server or browser computers. Thus, on a Microsoft Windows machine, it might be that the file is actually located at C:\Front Page Web\Web\Mathematics\books.html, but the URL directory specification would show only /Mathematics/books.html.

For each server program the administrator gives a file name that is to be used as the default page for each directory. Typically, the name chosen is index.html or default.html. Whenever a browser requests a URL that ends with the name of a directory, the server program looks in that directory to see whether it contains a file with the default name. If so, the server delivers that file to the browser.

Anything that appears after the file name is either a target within the file, or input to the program in the file. Targets are specified using a pound sign (#), while input to programs usually follows a question mark. We shall not discuss this further in this section.

Consider the URL <http://www.sci.wsu.edu/math/Calendar>. This tells the browser to use hypertext transfer protocol to contact a machine named www.sci.wsu.edu, and to look in a subdirectory of the document root called math/Calendar for a file named index.html (the default). The actual path to the file on that Unix server is then /pub/math/Calendar/index.html. It might be that the same machine runs a second server program that listens to port 8080. A URL for a file on in that document tree might be <http://www.sci.wsu.edu:8080/special/contents.html>. Again, the same machine might run an anonymous ftp server listening to port 22. A URL for a document available by that means might have a URL of the form <ftp://www.sci.wsu.edu/pub/math/ftpdocs/program.exe>.

Most people do not pay much attention to URLs as they surf the Web. However, we have all needed at some point to go to a specific Web site. We may do that in almost any browser by typing the URL into the text box on a toolbar near the top of the browser window, and then pressing the <ENTER> key.

Local Customization

By now you are fully aware that the entire idea behind using a browser is to do as much as possible locally. That idea continues into the presentation of the pages the browser retrieves. The browser allows you to control many aspects of the appearance of pages displayed in the browser, even to the point of overriding some of the aspects of the presentation put in by the page author. In particular, most browsers allow you to specify which fonts you want to use, and colors you prefer for the pages you view.

The Cache

The most time-consuming thing a browser does is retrieve pages over a network. Formatting and displaying those pages proceeds very quickly on the local computer, but waiting for those pages to arrive across the network occupies much of the time of the browser, and the person using it. To reduce the time required for that, virtually all browsers use file caches. The idea is simple: the browser stores every page it retrieves locally. Later, if the user wants to return to that page, or if an image from one page is used repeatedly on others, the browser can pull the file or image from its local cache, instead of having to get it over the network again.

This can sometimes cause problems for a browser user. Perhaps you want the original page from the server because it has changed. Perhaps your cache has grown so large that it is interfering with your computer's operations. In the first case, most browsers have a means of forcing the browser to reload from the server. For example, netscape permits you to hold down the <SHIFT> key while you click the reload button to force the browser to retrieve the page again. In the second case, you might need occasionally to clean out your cache. Doing so saves storage. You may also prevent the cache from becoming too large by controlling its permissible size. For example, in Netscape you can control the size through [Edit->Preferences->Advanced->Cache]

MIME

So many different forms of documents may be transferred over the Internet that browsers (and e-mail tools) need some means of keeping track of them. In particular, some documents need to be opened using a program not included in the browser, some files contain images in different formats, and some files should just be saved to a hard disk. Most browsers use two tools to sort the files out: the filename extension and the MIME type.

The filename extension is probably the more intuitive of the methods. Typically, filenames on Unix and Microsoft operating systems have the form `name.extension`. The name is whatever you want it to be, but the extension contains information concerning the nature of the contents of the file. For example, the browser would understand that a file named `bob.gif` contains a digital image in graphics interchange format, and should be displayed directly, while a file named `bob.ps` contains a document in the Adobe postscript language, and should be passed to an external viewer such as ghostview. The extensions are usually kept short - indeed, in 16-bit Microsoft operating systems the extension was required to be at most three characters. On Unix machines and 32-bit Microsoft operating systems the extension may have any length, but it is very rare to find an extension longer than four characters.

On the other hand, filenames can lie. You may put your postscript document or GIF image into files with any names you want. For example, if your browser receives a file with the name `bob.postscript`, it probably will not know how to handle it. It was for this reason that MIME (_____) was created. MIME provides more direct information about the content of files passed over the Internet, independent of the names of those files. Most Web and email servers are trained to send MIME information, and Web authors may do so explicitly in their documents as well. MIME types consist of two words separated by a slash ("/"). For example, the GIF image

we transferred earlier would have a MIME type of "image/gif". The postscript document would report a MIME type of "text/postscript". The browser, in turn, either knows by default or may be trained to recognize these MIME types and to know how to handle them.

Interaction and Animation

In the beginning of the World Wide Web it was envisioned that documents would be transferred together with easy ways to retrieve related information. However, it was not very long at all until it became clear that browsers needed to run programs. Web site managers wanted to receive information from people viewing their site, or they wanted to provide means for viewers to interact with their displays. Sometimes they just wanted to make their pages look more lively. Regardless of their motivation, any kind of animation or interaction requires that a program run somewhere, and that program needed to be triggered by the browser of the viewer.

Many ways of allowing browsers to run programs were developed, which may be grouped into "server-side" programs and "client-side" programs. As you might imagine, server-side programs run on the server machine, at the command of the browser. The most common examples of such programs are Common Gateway Interface programs and Java servlets. These are used primarily when the author of the page being viewed wants to receive and record input from the viewer of her pages.

Client-side programs run either directly in the browser, or are interpreted by some program outside of the browser. For example, Java programs run in most popular browsers directly, while Maple worksheets require the program Waterloo Maple to be started by the browser, whereupon the Maple program runs the input file that was retrieved over the Internet.

Running programs is a critical part of scientific use of the Web, but it is also fraught with danger. Any time you allow someone to run a program on your computer, you take a chance that he will do something you don't want to your machine. Various different client-side programs attempt to protect you in a variety of ways, which we shall detail later. Suffice it, for the moment, to say that you do not have to allow programs to run on your computer. All browsers must allow you the option not to run Java, Javascript, and other such programs automatically, and many cautious users do this. Server side programs do not present the same threat.

This topic is large and important in scientific use of the Internet, and we shall discuss it at length in later chapters.

EXAMPLE: SALMON GROWTH

Compensatory Growth in Salmon

Abstract: Salmon, like many animals, can adopt different strategies in the face of conditions that are not optimal. This study compared the growth of salmon undergoing stress (inflicted by the researchers) with that of a control group. The salmon in the stressed groups were able to adopt strategies that compensated in part for the privations they endured. The format of the experiment was to treat the groups of salmon in different ways for thirty six days, and then to mix the groups and see how they competed.

Reference: Ecology 78 (8), 1997, pp. 2385-2400. Data come from Figure 4.

Data Description: The data represent upper bounds on the masses of salmon at various times in the study. Day zero refers to the end of the period in which the salmon were manipulated.

Data (grams):

Day	Control	Rest. Food	Low Temp.
-36	4.2	4.3	4.3
0	8.5	6.1	4.7
4	8.6	7.3	4.9
16	9.4	8.0	5.6
26	9.8	8.8	6.0
49	10.3	9.2	7.1
80	11.2	9.8	7.5
208	15.5	14.8	12.5

1. Discuss the data from the experiment.

Below is a part of a Web page that gives an exercise intended for undergraduate science majors. The exercise itself concerns a study from the literature of compensation for adverse conditions in salmon growth. The page itself has a few interesting features.

First, the title appears in an unusual font and color. The horizontal rule underneath it is again different from the default. Finally, all of the text beneath the horizontal rule is set as a table, with item names on the left in a green font, and data and descriptions on the right. You may view the entire page at <http://www.sci.wsu.edu/idea/Math300/salmon.html>. We will use this page as an example for all of our Web typesetting.

2.3 Using a Browser

Nomenclature

Throughout this text we use certain common terms in a technical sense. The terms, and any actions they describe, are listed below.

Select -- whenever we want to apply some formatting to a block of text we must first mark that text in some way. The way we do that is by using the mouse to click the uppermost left edge of the text block, and then holding the mouse button down while we drag it to the bottom right edge of the text block. As we do this, the text should change color to white, in the background should change color to black, highlighting the text we have selected. After that we can apply whatever format specification we wish to the highlighted text.

Keystrokes -- we indicate keystrokes by enclosing them in angle brackets. Thus, if we want you to push the "Enter" key, then we write "press <Enter>".

Menu -- most computer programs now make use of menus. The menu is usually a bar near the top of the window for our application. It typically contains items labeled "File", "Edit", and so on. When we refer to menu items we will use the notation [Item→Inner Item→Inner Inner Item] to indicate the progression of menu items we should click. For example, in most applications we can save a file by clicking [File→Save]. This notation indicates that we use the mouse to click the File menu, and then we click the Save item within that menu.

Pixel - a Picture Element. This is a unit of measurement for objects on a computer display - basically the size of one dot of white on your screen.

Wysiwyg - an acronym for "What you see is what you get". Typically in typesetting text of any kind, there are several ways to look at the document: some show the typesetting commands, and others show only the finished product. When wysiwyg editors appeared in the decade of the 1980's they were revolutionary - typists could see the product before they printed it. Now the point is that we want a wysiwyg HTML editor.

2.4 Editing Web Pages Using a Browser

Many modern browser programs have HTML editors built into them. Typically, one loads a page in the browser, and then can open that same page in a wysiwyg editor. The editor shows some approximation of what the page will look like when displayed on a browser, and provides toolbars and menus to insert the HTML items you need. We'll create a page identical to the one shown in Section 2.3 to illustrate the process.

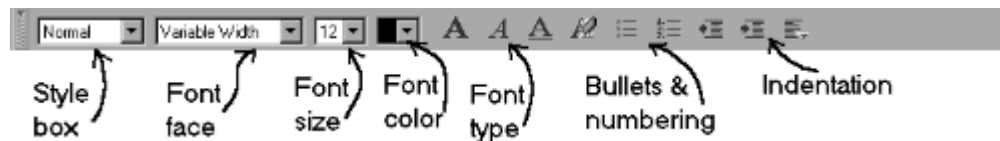
HTML editors are very similar, and indeed, look very much like word processor editors of any kind. Typically, there is a menu bar at the top of the editor window featuring the usual suspect for menu items: File, Edit, View, Insert, Format, and so on. Below that we find a toolbar that holds buttons to provide shortcuts to frequently used functions. Below that lies another toolbar containing font specification boxes and paragraph formatting shortcut buttons. Finally, below that we find a window where the text we type appears. This description applies to most of the popular

editors, including Netscape Composer, Microsoft Front Page, Microsoft Word, and others. Since these are so similar, and since Netscape Composer is free and platform-independent, we use that editor to develop our example page.

First, we open a blank page. To do that we start Netscape, and then click [File->New->Blank Page]. We should see a new window that satisfies the description given above. All we have to do is fill it up now. We could start with the last lines of the page, but we won't. The title is "Compensatory Growth in Salmon". We type that into the screen. Obviously, there is no formatting for that text yet.

We could do the formatting simply by picking a font family and size. However, that approach is fraught with problems. What if we are creating the page on a Windows machine and our viewer is using a Unix machine? Our viewer doesn't have an Arial font, and so she sees a typewriter font instead. What if our viewer is not using Netscape, but instead is using some unknown browser program that handles font tags differently, or not at all? We want to be sure that this title looks like title on all browsers that load this page. What if we specify a small font, but our user has made all of his fonts small to make the most of a small screen? In that case, our fonts might be illegible. Fortunately, HTML comes with tags that are supposed to represent headers, and which are used uniformly by all browsers. We will render this title text using the largest header format.

The header formats may be selected in the list box at the top left of the window. Usually, when Composer starts, this contains the word "Normal". Make sure that the cursor is somewhere in the text you just typed, and then click the little arrow at the right of that list box to see a list of all the format tags from which you may choose. Then click "Heading 1". The type for the title should become much larger. If not, check again to be sure that you had the cursor placed somewhere in "Compensatory Growth...".



We may change the color of the title text by selecting all of it, then clicking the font color selector tool on the toolbar. This is the small list box fourth from the left on the second tool bar. Click the small triangular button to display a palette of colors from which to choose. We chose blue, but you may choose your favorite from among those colors.

The title is still justified left. We want it in the center. To do that, again click in the center of the text and then click [Format->Align->Center] from the menu. Your title should now be centered on the page.

Now we have a title in a large font in our favorite color. Next we insert a horizontal line. Move to the line after the title, and click [Insert->Horizontal Line] from the menu, or click the little button labeled with a horizontal line from the topmost toolbar. The line that appears should span the entire width of the page, and is very thin. We prefer a line that only spans 80% of the page, is centered, and is thicker. To change the nature of this line, right-click on the line. A menu appears inviting us to change any of several things: click "Horizontal Line Properties". A dialog box appears allowing us to control the appearance of the line. Enter 10 in the box labeled "Height", and enter 80 in the box labeled width. The width setting should already be given as a percentage of the window, but note that we could change the specification to pixels in the box to the right of the width specification. The height specification is not a percentage of the screen height, but instead is measured in pixels. Note also that there is a radio button where we could change the alignment of the line to the left or right. It should be in the center by default. When you are finished, click the OK button.

Next, put in a couple of blank lines by pressing the <ENTER> key a couple of times. We should note that this does more than you might think. The browser actually ignores blank lines, so in order to get one, you must put certain HTML tags into your document. Pressing <ENTER> starts a new paragraph.

Now we are ready for the body of the document, which is entirely composed of a table. It may not look to you like a table, but observe that all of the labels on the left are aligned, and the text and smaller table on the right are likewise aligned. This may only be done with a table. Click [Insert->Table->Table...] to produce a dialog box where you may enter various characteristics of the table you want. First, select the text box at the top of the screen labeled "Columns", and specify two columns. You may put in the number of rows you want, but it is not important - we can create new rows very easily. In the area below the column specification, click the radio button to center the table on the page. Below that there is a check box and other text boxes that control the borders. The border check box should be checked by default - click the box to "uncheck" it, so that we do not have a border around our table. Further down, there is a check box indicating whether we want to have columns of equal width. Uncheck that also, so that the left column can be narrower than the right. When you are finished, click the "OK" button.

We are now ready to put the content in our table. Click in the upper-leftmost cell of the table, and type "Abstract:". Observe that the left column grows wider as you type. That is because you unchecked the "Equal Column Widths" box in the table dialog. To move to the next cell to the right, just use the <TAB> key. Type some of the abstract as given in the example above: "Salmon, like many animals...". When you are tired of typing, you may move to the next cell using <TAB>. Perhaps you made a table with only one row, because you believed us when we said it was easy to make more rows. In that case, there is no next cell. That's no problem: just hit <TAB> anyway, and Composer makes a new row for you. Your cursor will be in the leftmost cell of that row, where you may type "Reference:".

In the figure from Section 2.3, we notice that the entries in the left column of the table are in green font. It would be nice if we could change the entire column to use a green font, but we can't. Instead, we need laboriously to select each label in the left-hand column, and then click [Format->Color] to obtain a palette from which we may choose our favorite color for the font. When we learn about Cascading Style Sheets later in this chapter, we should be able to do this formatting more simply.

At some point, you will need to make a change to the format of the table. You may do that any time by clicking with the right mouse button inside the table. A menu appears in which you may click "Table Properties". You will notice that the Table dialog box actually has three tabs (see them at the top of the page) labeled "Table", "Row", and "Cell". You may control the text alignment, background color, and other properties using these three tabs. You undoubtedly wonder why there is no "Column" tab. The reason is that tables in HTML are organized by row, and there is no HTML tag to allow control of entire columns together. You might consider this to be one of the many weaknesses of HTML - we would agree.

We should note here also that you may insert rows and columns using some of these same features. When you right-click in the table, a menu appears. This is the same menu that allowed us to click "Table Properties" before. To add rows or columns, click "Insert" and then click the menu item that corresponds to the thing you want to insert. The inserted object appears after the one where the cursor is. In other words, the inserted row appears below the current row, and an inserted column appears to the right of the current column.

The next feature of note in the page is the table that appears in the row labeled "Data". That is, in fact, the second table on the page, and it lies inside the main table. This is not a problem: we put our cursor in the rightmost cell of the "Data" row, and again click [Insert->Table->Table...]. This time when the table dialog box comes up, make four columns, and leave the check box for the border checked.

Both of the table dialog boxes - the one you get from [Insert->Table->Table...] and the one you get on right-clicking the table - contain options for controlling the table borders, cell spacing, and cell padding. The border specification pertains to the outside visible borders of the table. If you set it to zero, that is the same as forcing the table not to have a border. If you set the value to e.g. 4 or 8, then the outside border of the table will be thick, but the inside borders between cells should not be affected. Those spaces between the lines dividing cells are controlled by the cell spacing. The cell padding specification applies to the white space between the content of a cell and the cell boundary. In other words, if you set the cell padding to 10 pixels, then there will be a lot of space around your text within a cell, while if you have no padding, then your text should push right up to the edge of the table cell. Try putting in various different values for these settings, and note the differences.

Type the header row for the data table, and then one row of data. Note that the header row is supposed to look different than the data rows. You can change some characteristics of the entire header row using the table dialog box. Unfortunately, you cannot change character properties for an entire row - to get a bold face font, you must again go through the tedious process of selecting the text in each cell and using [Format->Style] or a toolbar button to change the face. Use one of those methods to change the fonts in the header cells to bold face.

The last feature of the page is a list of questions at the bottom. The questions are in a numbered list. The numbers in the list are the result of the action of the browser program - not the HTML programmer. Make a cell in your larger table to put the list in, and then click the button near the right end of the toolbar that shows numbers with horizontal lines to their right. A pound sign (#) should appear. When the browser sees the HTML tag that corresponds to that, it will choose a number to put there. Type the first question. When you are finished, press <ENTER>, and a new pound sign will appear. You may repeat this process as needed.

If you started typing the question before you clicked the number to do the enumeration, have no fear. Just select the paragraphs you want to be numbered, and then click the number button on the toolbar. Composer will interpret each new paragraph as a line to be numbered. If you should make a mistake and number a line you don't want to appear as part of the list, then click in that line and then click the number button again. The numbering will be removed.

To save your work on the local computer, click [File->Save]. The menu item [File->Publish] saves your work on a remote computer. To use the latter, you will need to specify the name of the remote computer, and the details of your account there. See the discussion of ftp in the previous chapter.

To reiterate, this discussion used Netscape Composer for its instructions, but the kinds of commands found here are almost universal. The button labels may change, the details of the dialog boxes may be different, but for the most part, the specifications will be similar.

Netscape Composer Commands

Here is a brief synopsis of some of the most common tasks in Netscape Composer, and a very brief discussion of how to perform them. It is not intended to be exhaustive.

Boldface - select the text you wish to be in boldface, then click the button on the second toolbar that shows the letter A in a bold face. Alternatively, you may also select the text and then click [Format->Style->Bold].

Colors - To change the background or text colors on your page, use the menu item [Format->Page Colors and Properties]. This brings up a dialog box in which you may set the colors. For example, to change the background color for your page, first click

the radio button labeled "Use custom colors". Next, click the button below it labeled "Background...". Select the color you like best from the palette provided. You can use analogous procedures to change the colors for text, links, and so on.

Heading - Click the list box at the left of the second row of tool bars - ordinarily this has the entry "Normal" in it. You should see a listing of styles to use. Choose a heading style to your liking. "Heading 1" is a larger font, "Heading 5" is a smaller font. If you have not already typed the heading, then typing now will produce text in that heading style. If you already typed the text you want to be in the heading, then select it before you use the Heading list box.

Horizontal rule - click the button on the toolbar indicating a horizontal rule. You may also click insert horizontal line from the menu.

Images -- to insert an image click the button on the first toolbar that shows a small rectangle with three colorful objects inside it. This brings up a dialog box in which you are required to enter several pieces of information. The first is the name of the file containing image. If you not created such a file that you cannot put the image in your page. If you have a file then you may either type its name in the box provided at the top of the dialog box, or click the button labeled "Browse...", after which you may click to the proper directory and file that you want. It may be that this is all is required, however if you want special formatting for the image, or other alternatives, you must enter them here. A discussion of some of the options appears later in the text.

Italics - select the text you wish to be in italics, then click the button on the second toolbar that shows the letter A in a slanted face. You may also use the menu [Format→Style→Italic].

Itemized list - click the button on the second toolbar showing three so-called bullets (•) followed by three horizontal lines. After this, every time you press <Enter> you'll get a new bullet.

Links - Of course, the real power of the Internet is in allowing links to pages and other sites. If you want to use one in your pages, first type the label that will represent the link on your page. Then select that label text, and click on the button with the picture of a chain link on the first toolbar. This opens a dialog box in which you must enter link information. If the file is local to the machine on which you are working, then you can use the button labeled "browse file" to find a way to the file and enter it. If it is not local, you must type in the actual URL, e.g. <http://www.mysite.com/fun.html>. When you're finished, click the OK button.

Numbered list - click the button on the second toolbar showing the numbers 1 2 and 3 followed by horizontal lines. After that, every time you press <Enter> you'll get the next number in the sequence from which you are working, where you can type another line.

New Page - Click [File→New→Blank Page].

Save - Click [FileSave]. If it is the first time you have saved the file, this brings up a dialog box in which you must enter the file name. The file name should always end in ".html" or ".htm".

Table - To insert a table at the cursor, click the button on the first toolbar with an image of a table on it (two rows, three columns). This brings up a dialog box where you may specify attributes of the table. The only critical thing to specify here is the number of columns - you can always add rows later.

Text (no special formatting) - Just type.

Underline -- select the text you wish to appear with an underline, and then click the button on the second toolbar that shows the letter A underlined. You might choose instead to use the menu item [Format→Style→Underline].

2.5 Hypertext Markup Language

From the inception of the World Wide Web, it was clear that there was a need for a language for the documents to be transferred that would be common to all browser programs. The language needed to satisfy several important criteria.

- It needed to be text-based; no files with non-printing characters would do.
- It needed to be simple, since there were no sophisticated editors with point-and-click interfaces to help the authors of the time.
- It needed to allow for variations in the way browsers chose to present the styles and sizes it specified - in particular, it could not be too choosy about fonts.

There were models for such languages already in existence at that time: they were called *markup languages*. A markup language is basically any language that satisfies the criteria above: it is purely based on text, and mixes human-readable formatting instructions with the content text. The two prominent examples of markup languages of that time were called TeX and SGML. TeX was a language developed at the University of Berkeley by a group headed by Donald Knuth. It and its variants constituted a defacto standard for mathematical typesetting by the late 1980s, and indeed, remains the standard today. Unfortunately, TeX is not particularly simple, nor does it allow for the kind of variation that would be required for screen presentation of documents on different kinds of browsers on different operating systems. TeX is quite picky about fonts. SGML suffered from many of the same problems.

These criteria and the need for something simpler than the extant markup languages led early researchers to develop HyperText Markup Language, known by its acronym: HTML. HTML was largely derived from SGML, but had a severely limited command set for simplicity. This made files more compact and documents easier to write. It later also led to a number of regrettable proprietary developments of markup text; but that is another story.

We have already seen that there are many ways of creating HTML code now without knowing any of the actual language, but there is much on the Web that does not make much sense unless one knows HTML. Moreover, it is difficult and inadvisable to write pages for the Web using only a proprietary HTML editor, due to the tendency of those editors to be too platform-specific in their use of fonts, and other formatting instructions.

The basic rules for HTML are simple

- Commands in HTML are framed in "angle brackets". The angle brackets are, in fact, the "less than" and "greater than" symbols on the keyboard. Thus each command in the text appears as e.g. `<COMMAND>`.
- Commands in HTML are frequently called "tags". The tag is the entire structure comprising the angle brackets and command name.
- Each HTML tag initiates a change in formatting that remains in effect until the tag is closed. Thus, each HTML directive really has three parts: the tag that initiates the directive, the content to which the formatting applies, and the tag that closes the formatting directive. Closing tags look almost like the opening tags, differing only in having a slash "/" between the left angle bracket and the command name. Thus, a paragraph begins with the tag `<P>` and ends with the tag `</P>`.
- Often it is possible to leave off the ending tag, since the browser can figure out what is meant by the context. For example, if you type `<P>` to begin a paragraph, and then use the `<P>` again to begin the following paragraph, then the browser can interpret those tags as denoting two separate paragraphs, because it knows that paragraph tags cannot nest. While omitting closing tags is common, it is considered bad style.

EXAMPLE

We consider the HTML required to create the salmon page. The first step is to start a text editor. You say you have never used a text editor before? We are talking about a program like notepad.exe in Windows, or vi or emacs on Unix machines. All we need is a program that allows us to enter text directly into a file. To start notepad, click [Start->Run], then type "notepad.exe salmon.htm" into the resulting command box, or click [Start->Programs->Accessories->Notepad]. To start an editor in Unix, type "vi salmon.htm" or "emacs salmon.htm" on a command line. Of course, there are many other ways to run text editors, and many other editors from which to choose.

Once the editor is running, you can begin to enter HTML text into the file. The first tag should be the <HTML> tag. This simply alerts the browser to the fact that HTML is coming, as opposed to text in some other language, or a binary file. You will need a matching closing tag for the <HTML> - you may as well put it in now. It looks like </HTML>.

Next, you should put in some header information. HTML provides a <HEAD> tag for this purpose. Anything between the <HEAD> and its closing tag does not show on the browser screen. Again you may as well put the closing tag in now, so you won't forget.

Between the header tags you put information about the document, such as its title, the program that was used to create it, the name of the author, and so on. HTML provides a couple of tags to help with this. Type in a title entry that looks something like <TITLE>Salmon Growth Compensation</TITLE>. The title that appear here will not be seen by the reader of the page, but is the title that will be reported by all the search engines that list your page. You might also want to put in author information. You could use a <META> tag for that purpose, it might look like <META NAME="Your Name" CONTENT="Your Address">. The <META> tag provides a means for you to provide any kind of non-printing information to the reader. The reader does not see it directly in the page presentation, but can look at it if she wishes by clicking [View->Page Source] (or its equivalent) in her browser.

It is time to put something into the file that actually shows on the screen. Below the header information, that is, after the </HEAD> tag and before </HTML>, you should type in a beginning and ending <BODY> tag. The body tag controls the basic look of the page, including the background and foreground colors, colors for linked text, and background images, if any. To make a point, let's make our screen yellow. We put in a tag that looks like <BODY BGCOLOR="yellow">. Note that the color specification may be entered using the classic RGB format: for example we could have entered the color as "ffff00". See the section concerning color specifications. Fortunately for us, most browsers also understand a certain number of stock colors, such as red, blue, and yellow.

Now the actual salmon page had a white background, so change the yellow specification above to "white". We are ready for some actual text. HTML provides a number of tags for headings of various sizes. We will use the largest one as we did in the previous section. Type a tag of the form <H1 ALIGN="center">Compensatory Growth in Salmon</H1>. This puts a visible header line into the document, centered on the screen. Unfortunately, it uses the default font and color for that header. If we wish to see the header in blue, as it appears on our page, we must insert one of the most dreaded tags in HTML - a tag.

The tag is much maligned among those in the know regarding HTML, for good reason. The tag is used to specify a font face, color, or size. As such it provides no information about the structure of the document, but instead only issues machine-dependent instructions for the presentation of the material. As such, it presents a variety of problems for the browser. It may specify a font face that the operating system does not support, or a size

that is too small to read on a given machine. On the other hand, in the early 1990s the tag was the only option for those who wanted greater control over the appearance of their documents, and it still provides a quick and very dirty way of assigning certain font characteristics. What we need here is a tag inside the <H1> tags to specify the color. Then entire entry now looks like the following line.

```
<H1 ALIGN="center"><FONT color="3333ff">Compensatory Growth in Salmon</FONT></H1>
```

This illustrates nicely the way that one may nest tags in HTML. The properties of the inside tags simply add onto those outside. On the other hand, you should use care not to close the tags in the wrong order, because that can cause unpredictable things to happen. One should also get used to nesting tags properly because XML is less tolerant of author error than HTML, so in the future, what is now simply good style will become essential.

The heading is done - below it we want a horizontal rule. The tag to make that is <HR WIDTH="80%" size="12">. This tag is largely self-explanatory: HR stands for horizontal rule, which is to occupy eighty percent of the screen's width, and is to be 12 pixels high.

Now we move down a few lines. All that is required is to put in a few empty paragraphs or line breaks. It is worth noting the difference. Good style dictates that the paragraphs should be typed as <P></P>, while the line breaks just appear as
. Indeed, there can be little reason to close a line break specification, which makes it unique among HTML tags. The difference between the two tags is small: the paragraph tag leaves a little space above it in addition to putting in a line break; the
 tag only puts in a line break. Choose whichever suits your fancy.

The meat of the page is the table containing all of the information. As we noted in the previous section, there are actually two tables. The outside table formats most of the page, while the inside table actually fills one cell of the outside one. Only the inside table has borders. To insert a table, we use the <TABLE> tag. Since it is an extended construction, put in the entire first row of the table thus:

```
<TABLE WIDTH="100%" BORDER=0>
<TR>
<TD><FONT COLOR="#009900">Abstract</FONT>:</TD>
<TD>Salmon, like many animals, can adopt
different strategies in
the face of conditions that are not optimal.
This study compared the growth
of salmon undergoing stress
(infllicted by the researchers) with that of
a control group.
</TD>
</TR>
</TABLE>
```

You can see immediately that the <TABLE> tag allows some of the same options that other tags do, including the width specification (as with the <HR> tag). It also allows the <ALIGN> option, and others we shall see shortly. Within the table, formatting is controlled by the table row (<TR>) and table data (<TD>) tags. In other words, the table is organized by rows, and the contents of cells within rows are enclosed in <TD> tags. Each of these tags takes similar

options to the table tag itself; that is, one may specify width, borders, padding and alignment for rows and cells. To add a new row, simply put in opening and closing <TR> tags, with the proper number of <TD> tags, and keep going.

In the third row there is a second table embedded in the main table. This one has visible borders. The tag to use this time is <TABLE BORDER=2>. The <TR> tags are the same, but there are now four <TD></TD> pairs. Note that this table must be nested entirely within a <TD></TD> pair for the larger table - in other words, it is entirely within a single cell.

This table-within-a-table has header text in its first row. The header text is in a bold face. To get bold lettering in any font in HTML, enclose the text in a pair. Using <I> instead gives an italic face, <TT> produces a teletype or typewriter font, while <U> gives an underlined text decoration.

In the bottom rightmost cell of the main table, we see a numbered list of questions. There are several ways we could create such a list, but HTML provides an easy way in which we don't have to count the items ourselves. We use the tag. OL stands for *ordered list*. Each item in the list is set apart within list item tags: . Thus, the list in the last cell looks something like the following.

```
<OL>
```

```
<LI>Use a least squares process to fit linear and quadratic polynomials to each set of data</LI>
```

```
<LI>What is the significance of the coefficient of the first degree term in the linear fit? What is its significance in the quadratic fit?
```

```
<LI> ... </LI>
```

```
</OL>
```

There are other kinds of lists available in HTML. For a list that is not numbered, but instead has "bullets" at the head of each line, use (unordered list) together with the same tags. For a list with no numbers or bullets at all, use a definition list - i.e. <DL> tags. In this case, however, list items are designated using the <DT> (definition term) tag.

At this point, we are finished. A little bit of text outside the table at the bottom of the page gives author information, and then we have the closing </BODY> and </HTML> tags. The entire listing may be found below.

```
<HTML>
```

```
<HEAD>
```

```
    <TITLE>Compensatory Growth in Salmon</TITLE>
```

```
</HEAD>
```

```
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000EF" VLINK="#51188E" ALINK="#FF0000">
```

```
<CENTER>
```

```
<h1>
```

```
<font color="#3333FF">Compensatory Growth in Salmon</font></h1></CENTER>
```

```
<hr SIZE=12 WIDTH="80%">
```

```
<br><br>
```

```
<TABLE WIDTH="100%" BORDER=0>
```

<TR>
<TD>Abstract:</TD>
<TD>Salmon, like many animals, can adopt different strategies in the face of conditions that are not optimal. This study compared the growth of salmon undergoing stress (inflicted by the researchers) with that of a control group. The salmon in the stressed groups were able to adopt strategies that compensated in part for the privations they endured. The format of the experiment was to treat the groups of salmon in different ways for thirty six days, and then to mix the groups and see how they competed. </TD>

</TR>
<TR>
<TD>Reference:</TD>
<TD>Ecology 78 (8), 1997, pp. 2385-2400. Data come from Figure 4.</TD>

</TR>
<TR>
<TD>Data Description:</TD>
<TD>The data represent upper bounds on the masses of salmon at various times in the study. Day zero refers to the end of the period in which the salmon were manipulated.</TD>

</TR>
<TR>
<TD>Data (grams):</TD>
<TD>

<TABLE BORDER WIDTH="100%" NOSAVE >
<TR>
<TD>Day</TD>
<TD>Control</TD>
<TD>Rest. Food</TD>
<TD>Low Temp.</TD>

</TR>
<TR>
<TD>-36</TD>

<TD>4.2</TD>
<TD>4.3</TD>
<TD>4.3</TD>
</TR>
<TR>
<TD>0</TD>
<TD>8.5</TD>
<TD>6.1</TD>
<TD>4.7</TD>
</TR>
<TR>
<TD>4</TD>
<TD>8.6</TD>
<TD>7.3</TD>
<TD>4.9</TD></TR>
<TR>
<TD>16</TD>
<TD>9.4</TD>
<TD>8.0</TD>
<TD>5.6</TD>
</TR>
<TR>
<TD>26</TD>
<TD>9.8</TD>
<TD>8.8</TD>
<TD>6.0</TD>
</TR>
<TR>
<TD>49</TD>
<TD>10.3</TD>
<TD>9.2</TD>
<TD>7.1</TD>
</TR>
<TR>
<TD>80</TD>
<TD>11.2</TD>
<TD>9.8</TD>
<TD>7.5</TD>
</TR>

```
<TR>
<TD>208</TD>
<TD>15.5</TD>
<TD>14.8</TD>
<TD>12.5</TD>
</TR>
</table>
    </TD>
</TR>
```

```
<TR>
<TD><font color="#009900">Analysis</font>:</TD>
<TD>
<OL>
<li>Discuss the data from the experiment.    </li>
<li>Fit quadratic polynomials to each data set for each year of the
study.    
What is the physical significance of the value of each
coefficient?</li>
<li>Can you formulate a functional form that describes the data more
effectively
than the quadratic?     If so, fit that to the data and discuss the
results.</li>
<li>Compare the fitted functions for each group, and describe the
differences
between the groups in terms of the coefficients of those
functions.</li>
<li>The data seem to have a couple of inflection points.    
Speculate on
why that is.</li>
<li>The lower bounds of the masses on all three groups do not display
nearly
the differences that are seen in the upper bounds given above.    
Speculate
on why that is.</li>
</OL>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

HTML Tags

<A> This is the "anchor" tag, also known as a link. It allows the user to click an area of text or an image to call up a different page.

Attributes:

HREF="The URL of the page to which this link points."

Example:

The following link displays "WSU Mathematics Department" in the link color, and clicking the link will bring up a WSU Mathematics Department home page.

```
<A HREF="http://www.sci.wsu.edu/math/">WSU Mathematics Department</A>
```

**** Set the enclosed text in a bold face.

Example:

In this sentence, only the word **bold** is in a **bold** face.

<BODY> This tag controls the overall appearance of the page. It also encloses content that the viewer sees.

Attributes:

BGCOLOR="background color specification"

BACKGROUND="URL of background image"

LINK="link text color specification"

TEXT="text color specification"

VLINK="

ALINK="

**
** Insert a line break

Example:

This text appear on **
** two separate lines.

<DL> Create a definition list. A definition list does not include any bullet or numbering - instead it only uses indenting to indicate the items. See **<DT>** for an example.

<DT> Insert a term in the definition list. Each term has a **<DT>** tag.

Example:

```
<DL>
```

```
    <DT> This is the first item.
```

```
    <DT> This is the second item, on a separate line.
```

```
</DL>
```

**** This is rather like the **<I>** tag. It is used for emphasis.

<H?> This set of tags permits typesetting of a variety of header formats. The fonts used are graded in size from largest for the **<H1>** tag, to smallest for **<H6>**.

Example:

```
<H1>This is in a large font</H1>
```

```
<H2>This is set in a somewhat smaller font.</H2>
```

```
<H6>This might appear in a very small font.</H6>
```

<I> The enclosed text is set in an italic font.

Example:

One usually uses italics for **<I>emphasis</I>**.

**** This tag indicates a list item within either an ordered or unordered list.

Example:

```
<OL>
```

```
<LI>This is the first list item. In this case it will actually appear with a number in front of it.
```

```
<LI>This is the second item.
```

```
</OL>
```

**** Create an ordered list. In other words, the items in the list should be numbered consecutively. As the author of the page, you do not have to assign the numbers. You only need to designate the items of the list, and then the browser numbers them for you. See for an example.

<P> Insert a new paragraph. The effect of this is to insert a line break and leave some extra space above the new line. Indeed, the only difference between this tag and the
 tag is the extra space above the new line.

<TABLE> Insert a table.

Attributes:

BORDER="width of border in pixels". This number could be zero.

Example:

This is a three column table.

```
<TABLE BORDER=2>
```

```
<TR>
```

```
<TD>Left Column Heading</TD><TD COLSPAN=2>This spans two columns</TD>
```

```
</TR><TR>
```

```
<TD>First Item</TD><TD>Second Item</TD><TD>Third Item</TD>
```

```
</TR>
```

```
</TABLE>
```

<TD> This encloses a single cell within a table.

<TR> This encloses a row of a table.

<TT> These tags enclose a typewriter-style font. TT stands for teletype.

Example:

The Unix command to list details of files is <TT>ls -l</TT>.

**** Create an unordered list. In other words, the list items are headed by the small circles ("bullets"). See for an example.

2.6 Cascading Style Sheets

As the World Wide Web has matured, people have demanded more of it. Companies trying to do retail business over the Web find themselves managing sites containing thousands or tens of thousands of separate pages, each of which requires formatting for presentation. These companies want a uniform appearance for their pages, and want to be able to change that appearance at will. They need to create pages with the uniform style in a short time, and want to have complete control over the appearance of their pages.

In 1997, the W3C promulgated a proposed standard in response to this need. The standard was for Cascading Style Sheets (CSS)⁴. These are HTML pages that contain only formatting instructions. The idea is that we should be able to style our headings, text and other HTML constructs in any way we like without having to type those details in every single tag. Moreover, those style specifications should reside in a single file that all other HTML files might then refer to. In that case, a change to the style file would simultaneously change the presentation of every page that used it.

For example, suppose that we want our principal heading always to appear in sans serif font, in our corporate color, centered on the page. We may do that by using the following format everywhere we need an <H1>:

```
<H1 ALIGN="center">
<FONT COLOR="006666" TYPE="arial,Helvetica">
Our heading text
</FONT></H1>
```

There are several problems with this approach. Evidently it requires a good deal more typing. It requires us to put in an extra tag with the font specification every time a heading appears. Finally, if next week the company were purchased by an even bigger company which required us to conform to their Web standards, we would have to change those tags on each of the ten thousand pages we maintain to the following set of tags:

```
<H1 ALIGN="right"><FONT COLOR="0000A0" TYPE="times">
Our heading text
</FONT></H1>
```

Making these changes and others like them would require many person-weeks of work, thus adding to the cost of the acquisition.

Style tags address this issue by allowing us to set the formatting specification for any HTML tag in a single place. In the above example, we might put instructions at the top of each page. They would look like this:

```
<STYLE TYPE="text/css">
  H1 {
    text-align: "center";
    color: "0000A0";
    font-family: "times";
  }
</STYLE>
```

All of our heading lines would then look much simpler:

```
<H1>Our heading text</H1>.
```

4 [Http://www.w3.org/TR/REC-CSS1](http://www.w3.org/TR/REC-CSS1)

Now the tag shown above is only valid for the page in which it appears. We would prefer by far to be able to set styles across a large number of pages. We need a means of having one or more pages that define the styles for our site, which may be loaded from any page on that site. In short, we need the "LINK" tag. In the above example, we would put the STYLE specification above in some file, say "company_style.css". The "css" extension is standard for a style sheet. After that, we could gain access to the styles defined in that file by putting the following tag in all of our HTML files:

```
<LINK REL=stylesheet TYPE="text/css" HREF="company_style.css"
TITLE="main_style">
```

The LINK tag is largely self-explanatory. The critical item is the HREF attribute, which gives the URL of the stylesheet.

The TITLE attribute permits us to have more than one style sheet that applies to any given document. This is the reason for the term "cascading" that appears here. Each subsequent style sheet adds to the one before, or replaces its attributes, if there are duplicates.

The basic format for styles is simple: there are attributes that apply to individual HTML tags, and we enclose the attributes within braces, separated by semicolons. In abstract, style specifications have the following form.

```
HTML_TAG {
    attribute1: value1;
    attribute2: value2;
    ...
    attribute_n: value_n;
}
```

The colon separating the attribute from the value is obligatory, as is the semicolon at the end of the line. One may put quotation marks around the values if desired. The tabs at the beginning of each attribute line only contribute to the readability of the specification, and are optional. Thus, we could specify a particular paragraph style as follows.

```
P {
    font-family: sans-serif;
    font-size: larger;
    color: a00000;
    text-align: center;
    border: dashed;
}
```

This would force every paragraph in the document to be displayed in some kind of sans-serif font, in a larger size than the default, in a dark red color. The text would be centered on the page, with a dashed border around it.

An exhaustive list of attributes may be found at the W3C site: <http://www.w3.org/>_____. We also have an abbreviated listing of attributes in the example box below.

Style Sheet Properties

background-color - Possible values include a color specification or "transparent". This attribute applies to all elements, and is not inherited.

background-image - Possible values include a URL for the image, or "none". This attribute applies to all elements, and is not inherited.

border - This attribute is used to set and color border around any element. It is a shorthand notation encompassing the attributes border-top, border-right, etc. Possible values include a width, a style, or a color. The width could be a length or a percentage. The style specification could be "dotted", "dashed", "solid", or one of several other choices. Colors are specified as usual.

color - The value must be a color specification, as in "ff0000" for red, or "a000a0" for a violet. It applies to all elements. This attribute may be inherited. These specifications may also be made in the form rgb(i,j,k), where i, j, and k represent numbers between 0 and 255, inclusive. The number i again denotes the amount of red in the color, j the amount of green, and k the amount of blue.

float - Possible values include "left", "right", and "center". This is the means of aligning non-text elements on the page.

font-family - Possible values include a specific family, such as "Arial" for Microsoft machines, or a generic family, such as "sans-serif". More than one font family may be specified, separated by commas. The browser is then to look through its list of fonts for one that fits one of the members of the list. Thus "arial, helvetica" would allow a Netscape browser on a Unix computer to look for an arial font, fail, and then find an helvetica font. Both fonts would satisfy the generic appellation "sans-serif". There are only a few generic families defined: "serif", "sans-serif", "cursive", "fantasy", and "monospace".

font-size - Possible values include absolute sizes (such as "x-small", "small", "medium", "large", or "x-large"), or relative sizes (such as "smaller" or "larger"). It is also possible to specify the font size in an actual unit of length or percentage of screen size.

font-style - Possible values include "normal", "italic", and "oblique". It applies to all elements, and is inherited.

font-weight - Possible values include "normal", "bold", and "bolder". There are other settings in the form of numbers from 100 to 900. This attribute applies to all elements and is inherited. The attribute applies to all elements and is inherited.

height - The value must be a length. It applies to block-level elements, and is not inherited.

length units - There are many systems of units that you may use to specify length measurements. These include absolute units such as "px" - pixels; "pt" - typesetting points; "in" - inches; "mm" - millimeters; and "pc" - picas, where 1pc = 12pt. These absolute measurements should only be used when the physical characteristics of the output device are known - i.e for printing styles. For screen styles, one should use relative measurements such as "em" - the width the the letter M in the current font; or "ex" - the width of the letter X. Alternatively one might use keyword values such as "x-small", "small", "medium", "large", "x-large".

margin - Possible values length units or percentage units. The margin value applies to all four sides of the object in question: top, bottom, left, and right. If you need to set only one of these, you may use e.g. the "margin-left" attribute. This attribute applies to all elements and is not inherited.

percentage units - these always refer to some reference unit such as the width of the screen, or a standard font size. These allow relative specifications for all lengths with high precision. The format is always a number followed by a percent sign (%).

text-align - Possible values include "left", "right", "center", and "justify". It applies to block-level elements. This attribute may be inherited.

width - The value could be a length or a percentage of the width of the screen. This attribute applies to block-level elements. It is not inherited.

You will at some point want to have various different styles for text in your document. For example, you might want to emulate this book, and set most text in a larger sans-serif font, but have certain examples or illustrations in a smaller font, enclosed within a border. In short, the single paragraph specification above would not be sufficient. Instead, you would create several different paragraph styles. You do this by appending a period to the name of the element for which you want several styles, followed by names of the styles. For example, suppose we want a default paragraph style in a larger sized sans-serif font, but also want a second style in a smaller sans-serif font colored blue. The following style specification would do this.

```
<STYLE TYPE="text/css">
```

```
  P {
      font-family: sans-serif;
      font-size: larger;
  }
  P.smallblue {
      font-size: smaller;
      color: 0000a0;
  }
```

```
</STYLE>
```

```
<P>This is in the larger font.
```

```
<P CLASS="smallblue">This is the blue font, that actually appears in the
default size.
```

We see that if we want the larger font, we simply use the `<P>` tag, but if we want the smaller blue font, we use `<P CLASS="smallblue">`. Everything that is described in the "smallblue" paragraph style is relative to the parent paragraph style. Thus, the font for normal paragraphs is larger than the standard, while that for the "smallblue" paragraphs is actually the standard size. The "smallblue" paragraphs also receive their font family from the standard paragraph, so that they appear in a sans-serif font.

In short, we may specify as many different styles for any element as we wish, referring to them using the CLASS tag. The attributes of the parent element style are kept by the subordinate element styles, unless those attributes are set explicitly in a style declaration. For example, if we add a declaration of the form `P.serif {font-family: serif;}` to our style sheet, we have a new paragraph style using a serif font, in spite of the setting in the parent paragraph style. Note that you may not define subordinate classes of classes. The style specification `P.smallblue.serif` would not work.

3 OPERATING SYSTEMS

Hitherto we have discussed computer networks and browsers without considering the kinds of computers that these interact with. In reality, one can never entirely ignore the details of the local computer. Although the details of the hardware for the local computer do affect its behavior to some extent, most of what you see is largely governed by the computer's operating system. Operating systems come and go - there are many kinds extant, and many more that are now defunct. In this document we will focus on versions of Microsoft's Windows operating system, and on versions of the Unix operating system.

An operating system is a program that runs on a computer that is designed to do two things:

- manage permanent storage; and
- provide a structure for the user to control the computer.

Every operating system needs to provide a means for users to operate on the files they have in permanent storage. We need to create new documents, write new programs, and store image files. We also need to be able to organize those in some way, and to be able to modify the files, and run the programs. The operating system provides a framework in which we can do all of those things. Operating systems provide this functionality in a variety of ways.

- Some operating systems simply provide a command-line interface. In other words, the user simply types a command together with some options. This requires the user to know the name of the command and the labels for the options that modify the behavior of the command. In other words, this interface is more like using natural language to give someone instructions. It is thus less "user-friendly" - it requires more knowledge from the user. On the other hand, for one who knows the language, it is the fastest means of interacting with a computer, and the one that wastes the fewest computer resources on interaction with the user. Unix (without X windows) and DOS fall into this category.
- Some operating systems interact with the user entirely through a Graphical User Interface (GUI). This is the familiar point-and-click interaction featuring buttons and check boxes on the screen. The interaction is rather like taking a multiple choice test. The GUI offers several choices from which the user selects, which may lead to other "dialog boxes" with other choices. A GUI interface is much easier for computing beginners to navigate than a command-line interface. On the other hand, it offers much more limited choices for the user, uses more computer resources than the command line, and can be slower and more cumbersome for an expert to use. The Macintosh OS falls into this class.
- Most modern operating systems offer a mixture of the two interfaces above. A GUI offers a simple way for beginners to interact with the system, and offers experts an opportunity to look at several aspects of their work simultaneously. On the other hand, a command line is available for those in a hurry or those who need to script repetitive chores. Unix/X Windows, and all Microsoft Windows operating systems fall into this category. These are the operating systems we cover here, however, we shall not discuss the individual buttons and dialog boxes from the GUI tools in these operating systems. Instead, we will focus on the command languages, and on the manner in which the GUI is built onto the operating system.

The permanent storage on a computer is called a *file system*. The reason for this appellation is that the individual collections of information stored permanently are traditionally called *files*. The terminology arose in an obvious way: files are simply representations of documents, programs, and images. The operating system itself should provide a means for the user to create these blocks of information on the file system, and should further allow the user to move them, copy them, modify them, delete them, or transfer them over a network.

There are many different kinds of file systems, which usually correspond to differences in the operating systems. A short list of file systems with which you may be familiar, and the operating systems that use them, may be found in Table 1.

Table 1: File systems used by certain operating systems

Operating Systems	Filesystems
Windows 3.1	FAT16: FAT stands for File Allocation Table; 16 denotes the number of binary bits in the representation of addresses within the file system.
Windows 95/98/ME	FAT32: Since this uses a 32-bit address space, larger blocks of storage can be addressed.
Windows NT	FAT32 or NTFS: NT File System is superior to the FAT system.
Solaris, and certain other Unix	UFS: Unix File System
Linux	Ext2: Extended Unix File System

Users almost never need to know the details of the implementation of the file systems with which they are working, and we shall not discuss these further here. However, from time to time one encounters a disk that uses a file system different from the one native to the computer on which one is working. When that happens, it is important to recognize the problem. We will discuss later how to transfer files from one file system to another.

Virtually all file systems are seen by the user in a *directory* structure. A directory can constitute two separate things: it is an organizational tool for you, providing a number of different folders in which you may place your files; and frequently a directory might itself be the top node of a file system. Typically, a filesystem begins at a top node, which contains a number of directories, which in turn may contain more directories. In Unix, the top of the file system is called the root directory, and is labeled "/". In Windows operating systems, one might try to talk about a single top-level file system, but more reasonably one would say that there are many top level file systems labeled using letters of the alphabet followed by colons, vis c:, d:, e:, and so on. Typically, the file system on which the Windows operating system resides is c:. Historically, these file systems were directly linked to physical disk drives in Windows, so frequently they are referred to as the c: drive and d: drive, etc.. Today, they simply represent separate file systems on physical *partitions* of the disks.

Within the top level file systems lie many directories. These are labeled relative to the top-level file system using some delimiters to separate the different levels. In Unix, the usr directory that lies within the root file system is labeled "/usr". The bin directory within the usr directory is labeled "/usr/bin", and the X11 directory within that is "/usr/bin/X11". The label for a directory is called the *path*.

In Windows, the labels for individual directories are slightly different, but the principle is the same. The "Winnt" subdirectory of the c: file system is labeled c:\Winnt (in Windows NT). The System32 subdirectory of the Winnt directory is labeled c:\Winnt\System32. Notice that in the Unix paths the slashes slant forward, while in Windows paths, they slant the opposite direction.

The command used by both operating systems to specify the directory in which you want to work is "cd" - *change directory*. Thus, in a Unix operating system, if you want to work in the /usr/bin/X11 directory, you type the command "cd /usr/bin/X11". In a Windows operating system,

if you want to work in the System32 directory, you give the command "cd c:\Winnt\System32". After you give such a cd command, the directory it moves you to is called the current directory.

Each of the paths we have considered so far used the top-level directory as its reference. Such paths are called *absolute paths*. In Unix, an absolute path always begins with "/", while in Windows it begins with "c:\", "d:\", or something similar using a different letter. Directories and files may also be labeled relative to the current directory. Such labels are called *relative paths*. For example, suppose in a Unix file system that the current directory is /usr. Then the subdirectory X11 of the /usr/bin directory may be labeled using the relative path bin/X11, or using the absolute path /usr/bin/X11. Similarly in a Windows operating system, if the current directory is the top-level c:\ directory, then the System32 directory could be labeled using the relative path Winnt\System32, or using the absolute path c:\Winnt\System32. Note that in Unix an absolute path always starts with "/". A path that does not start with "/" is relative to the current directory. In Windows, an absolute path always starts with a letter and colon, or with a backslash ("\"), while any path that does not start in that way is relative to the current directory.

In both Unix and Windows operating systems, there are identical special labels for the current directory and for the directory that is the parent of the current one. In particular, the current directory is labeled using a period - ".", while the parent is labeled using two periods - "..". Thus, to change from the /usr directory to the top-level directory in a Unix operating system, you might type either "cd /" or "cd ..".

There is a summary of standard commands for interacting with the file system in Unix and Windows NT below. These are more or less universal, since the demands of file management are straightforward: one must be able to copy files, view them, delete them, and move them around. If the files contain executable programs, then one must be able to execute those programs. In the table, the commands are given in italic type. Optional arguments are given in brackets [], and a description of the arguments to the command is given below it.

<i>Function</i>	<i>OS</i>	<i>Command</i>
<i>Change Directory</i>	Unix	<i>cd path</i> Change to the directory given in the absolute or relative path <i>path</i> .
	NT	<i>cd path</i> Change to the directory given in the absolute or relative path <i>path</i> .
<i>Copy a file or directory</i>	Unix	<i>cp [-r -f -i] filefrom fileto</i> Copy the file named <i>filefrom</i> to the file named <i>fileto</i> . If <i>filefrom</i> is a directory and the -r option is used, then copying is done recursively - that is, the directory and all the files and subdirectories it contains are copied. The -f option forces the copy, without confirmation, while the -i option does the copy interactively, i.e. It prompts the user before copying any file. If <i>fileto</i> is a directory, then <i>filefrom</i> is copied to a file by the same name in the directory <i>fileto</i> .
	NT	<i>copy filefrom [fileto]</i> Similar to the Unix command. Note that if you leave the second argument blank, then <i>filefrom</i> is copied to a file of the same name on the current directory.

Function	OS	Command
List contents of a directory	Unix	<i>ls [-l -a -d] filename</i> List the filename if it exists. If filename is actually a directory, then this lists the contents of the directory. The -l option provides file details, -a lists all files including hidden files, and -d shows only the directory properties and not those of the files it contains.
	NT	<i>dir [/w /ah /ad] filename</i> Similar in function to the Unix command. The default is a detailed listing, akin to <i>ls -l</i> for Unix. The /w option shows a short listing. The /ah option shows hidden files, while the /ad option shows directory properties instead of those of the files.
Make a directory	Unix	<i>mkdir path</i> Makes a directory at <i>path</i> .
	NT	<i>mkdir path</i> Makes a directory at <i>path</i> .
Move or rename a file	Unix	<i>mv filefrom fileto</i> Moves the file currently called <i>filefrom</i> to the new name or location <i>fileto</i> .
	NT	<i>ren filefrom fileto</i> Moves the file currently named <i>filefrom</i> to the new name or location <i>fileto</i> .
Remove or Delete a file	Unix	<i>rm [-r -f -i] path</i> Removes the file <i>path</i> . If <i>path</i> is actually a directory and the -r option is specified, then all files and directories inside file are also removed. The -i option causes the user to be prompted before any file is removed, while the -f option forces removal without prompting.
	NT	<i>del path</i> Deletes the file <i>path</i> .
Remove a directory	Unix	<i>rmdir path</i> Removes the directory at <i>path</i> , if it is empty. If the directory still contains files, you must remove them first, or use <i>rm -r</i> .
	NT	<i>rmdir path</i> Removes the directory at <i>path</i> , if it is empty.

After we have control of the file system, we need to handle the individual files. Both operating systems that we are concerned with provide a number of rudimentary commands for examining

the contents of files, and adding to those contents without using an editor. We have summarized some of those commands in the next table.

<i>Function</i>	<i>OS</i>	<i>Command</i>
<i>Find files</i>	Unix	<i>find filesystem [-name -mtime] string -print</i> Find files in the file system <i>filesystem</i> having the attributes given in the option string. The option string can either ask for file names, or can look for files modified during a certain period of time. See the text for examples.
	NT	There is no corresponding command, but clicking Start->Find->Files or Folders provides an interface to a tool that actually encompasses several Unix commands, including <i>grep</i> and <i>find</i> .
<i>Find strings in files</i>	Unix	<i>grep "string" filename</i> Find the sequence of characters given in " <i>string</i> " in the file (or files) given in <i>filename</i>
	NT	There is no corresponding command, but clicking Start->Find->Files or Folders provides an interface to a tool that actually encompasses several Unix commands, including <i>grep</i> and <i>find</i> .
<i>Print a string on the screen</i>	Unix	<i>echo "string"</i> This just prints the characters in string on the screen.
	NT	<i>echo "string"</i> This just prints the characters in string on the screen.
<i>Find a command</i>	Unix	<i>whereis commandname</i> <i>which commandname</i> These two commands perform the same basic task in two different ways. Whereis looks for the command in a collection of standard directories, such as /bin and /usr/bin. Which looks for a command in the directories in the user's path.
	NT	No corresponding command.
<i>Get help with a command</i>	Unix	<i>apropos word</i> <i>man command</i> Apropos looks for all commands that use the word <i>word</i> in their help files. It is a good way to find commands that might perform the task you need to do. Man actually displays a help file for the command <i>command</i> . It shows the file one page at a time. You may view the next page by pressing the space bar. On some versions of Unix, the "page up" and "page down" keys work with <i>man</i> .

<i>Function</i>	<i>OS</i>	<i>Command</i>
	NT	<i>command /?</i> To get help with any command whose name you know in NT, you need only append the <i>/?</i> switch. If you don't know the command you need, then you may use the GUI help utility found at Start->Help

4 MATLAB

In this chapter we discuss several of the most popular computer programs for performing mathematical operations. Several such programs have established themselves in recent years. These all came from different roots, but have strongly overlapping capabilities now. Let's take a brief overview of the chief suspects.

The first program for doing mathematics using a computer was called Macsyma (Machine Sybolic Manipulator), and was developed at the Massachusetts Institute for Technology in the early part of the 1980s. It did what its name implied: limited manipulation of mathematical expressions. Initially it had no graphics capability. At the same time, a number of programs were being developed for non-symbolic aspects of mathematics and statistics. For example, Minitab was an early leader in statistical computing that could also plot data in a variety of formats. Later in that decade, other programs came onto the scene. By the beginning of the 1990s, several programs had assumed positions of preeminence in the world of computer mathematics: Matlab, Maple, Mathcad, and Mathematica. In the world of statistics, several other programs had become popular: in particular, S-Plus and SAS had drawn some of Minitab's market away. In this edition of the text, we shall not discuss these statistical programs any further.

Matlab was designed from the start as a means for manipulating matrices and vectors easily, with standard linear algebra tools, such as LU decomposition, built in. It only brought symbolic capabilities in later, and that by purchasing tools from Maple. Minitab is similar to Matlab, except in that the emphasis is on statistics instead of linear algebra. By contrast, Maple and Mathematica both followed in Macsyma's footsteps, and competed head-to-head. Mathematica was developed initially by the physicist Stephen Wolfram. Its syntax reflected, to some extent, the disciplinary bias of its author. Maple was developed by a consortium of mathematicians and computer scientists at the University of Waterloo in Canada, and its syntax thus followed more closely the traditional notation of mathematics. Mathcad grew differently, emerging as a free-form mathematical spreadsheet program that employed a more natural notation and a more extensive function library than comparable Microsoft and Borland products. Later, it also incorporated some symbolic capability derived from Maple.

In the beginning of the twenty-first century all of these programs can perform many of the same tasks, but some excel in certain applications. They can all solve simple systems of algebraic equations; they can all compute derivatives and antiderivatives of functions; they can all multiply matrices, and solve matrix equations; and they can all plot functions in a variety of ways. They are all capable of writing their output in a variety of formats, including HTML and TeX. On the other hand, most users would accept the proposition that Matlab is the best at handling numerical computations, while Maple and Mathematica are superior for symbolic calculations. Mathcad is marvelous for small-scale parametric modeling - the kind of "what if" calculations done for simple models or in the early stages of mathematical investigations.

4.1 Matlab Basics

The program Matlab dates from the mid-1980s. It came out of work by Cleve Moler, who remains the head of the company that produces it today. It was designed with relatively humble objectives: to provide a simple interface to high-quality numerical procedures for solving mathematical problems. Indeed, Matlab stands for Matrix Laboratory. It has effectively become the standard package for mathematicians and engineers who need to do quick-and-dirty estimates, simulations, designs, and tests of algorithms.

There are two basic versions of Matlab: the full version, and the student version. The student version is actually fairly complete, but does not have some of the plotting capabilities that the full version has. On the other hand, the cost of the student version is dramatically lower than that of the full version.

Starting matlab is simple enough. On a Windows machine or a Mac, simply double click the icon on the desktop or click Start->Programs->MATLAB->MATLAB. If you have the student version, then "MATLAB" is replaced by "Student MATLAB". On a Unix machine, it is even easier to start Matlab: simply type "matlab" in a command window. The result of any of these actions should be a window with a simple command prompt that looks like "»". This provides an opportunity to type. Try entering a simple computation such as "3+4", and press the <Enter> key. You should see

```
» 3+4  
  
ans =  
  
    7
```

This illustrates several basic notions behind Matlab. First, Matlab can do all basic arithmetic operations with a very natural syntax. Second, Matlab requires that every computation be assigned to a variable. If you do not specify the name of the variable, Matlab uses a default name: "ans". Matlab reuses the name "ans" as needed, so if you want to keep the result of a computation for reference later in a session, you should assign it to a different variable. For example,

```
» seven=3+4  
  
seven =  
  
    7
```

assigns the value 7 to a variable called "seven". We may view the contents of the variable `seven` at any time simply by typing its name.

```
» seven  
  
seven =  
  
    7
```

One may use variables interchangeably with the numbers or objects that they represent. For example, we could define a variable called "ten" by

```
» ten=seven+3
```

```
ten =
```

```
10
```

Matlab uses "+" to denote addition, "-" to denote subtraction, "*" for multiplication, "/" for division, and "^" to denote exponentiation. Thus,

```
» myvariable=2*(ten-2)+(ten/5)^2
```

```
myvariable =
```

```
20
```

Of course, arithmetic is not the only thing that Matlab can do. As we mentioned earlier, it was designed with linear algebra in mind. Thus, it excels in dealing with vectors and matrices. Vectors and matrices are defined using [brackets] to enclose them.

```
» row_vector=[1 2 3]
```

```
row_vector =
```

```
1    2    3
```

One may equally well separate the entries in the vector using commas:

```
» row_vector=[1,2,3]
```

```
row_vector =
```

```
1    2    3
```

To define a column vector, one may type the entries on separate lines.

```
» col_vector=[1
```

```
2
```

```
3]
```

```
col_vector =
```

```
1  
2  
3
```

If you prefer not to waste so much space in entering a column vector, you may separate the rows of the vector using semicolons. Semicolons always denote the end of a line of entries.

```
» col_vector=[1;2;3]
```

```
col_vector =
```

```
1  
2  
3
```

You might also get a column vector by taking the transpose of a row vector. Matlab uses an apostrophe (') to denote the transpose of any matrix or vector.

```
» col_vector=[1 2 3]'
```

```
col_vector =
```

```
1  
2  
3
```

One types matrices into Matlab analogously to the way one enters vectors. Indeed, Matlab does not distinguish between matrices and vectors - it treats vectors as $n \times 1$ or $1 \times n$ matrices. Thus we might enter a 2×3 matrix in the following way.

```
» A=[1 0 0  
0 1 0]
```

```
A =
```

```
1    0    0  
0    1    0
```

or alternatively as

```
» A=[1 0 0; 0 1 0]
```

```
A =
```

```
    1    0    0
    0    1    0
```

Matlab handles matrix arithmetic just as easily as scalar arithmetic. The symbols for the operations are the same, but their meaning is changed to that appropriate for matrix operations. For example the * symbol denotes an inner product.

```
» A*col_vector
```

```
ans =
```

```
    1
    2
```

Matlab keeps track of the dimension of the matrices it is handling, and refuses to perform illegal matrix operations. Remember that A is a 2 x 3 matrix.

```
» A*[1 2 3]
```

```
??? Error using ==> *
```

```
Inner matrix dimensions must agree.
```

By now you have noticed that Matlab always prints out the result of a computation. If you type only the name of a single variable, then Matlab prints the value of that variable. Sometimes you will not want to see the result of a computation. You may suppress output on any line by ending it with a semicolon. For example

```
» [1 2];
```

```
» [1 2]
```

```
ans =
```

```
    1    2
```

While numerical analysts will tell you that you should never compute an inverse (it is always more work than simply solving a matrix equation), Matlab has no problem in doing so. To compute the inverse of a matrix A we must invoke a built-in Matlab function called inv(). For example,

```
» A=[1 0
```

```
-1 1]
```

```
A =
```

```
    1    0
   -1    1
```

```
» x=inv(A)*[1 1]'
```

```
x =
```

```
    1
    2
```

Of course, if we had used the 2 x 3 A matrix that we had earlier, the `inv()` function would have given an error.

Note that Matlab is case sensitive. Thus, there are no functions called `Inv()` or `INV()`, and a variable called "a" would be different from A.

```
» A
```

```
A =
```

```
    1    0
   -1    1
```

```
EDU» a
```

```
??? Undefined function or variable 'a'.
```

There is no function to compute the dot product of vectors - it is not necessary. As we mentioned above, if x and y are column-vectors of the same dimension, then their dot product is found as $x' * y$ or $y' * x$.

When Matlab displays results, it uses four significant digits by default. It actually performs computations in double precision arithmetic, but displays the numbers using lower precision. If you should want to see more precision, you may use the long format. For example,

```
» z=2*pi
```

```
z =
```

```
    6.2832
```

```
» format long
```

```
» z
```

```
z =
```

```
6.28318530717959
```

```
» format short
```

```
» z
```

```
z =
```

```
6.2832
```

Observe also that Matlab has predefined values for a variable called `pi`, which of course is the famous trigonometric constant π .

The following example section summarizes some of the most useful Matlab commands.

Matlab Commands

The following are some of the simpler and more commonly used commands in Matlab. We have not provided details of their use. Indeed, most commands may be used in a variety of ways. See the help pages for more information.

abs(x) - returns the absolute value of the entries of a matrix x.

cond(x) - returns the 2-norm condition number of the matrix x.

det(x) - returns the determinant of the matrix x.

eig(x) - returns a vector containing the eigenvalues of the matrix x. `[v,d]=eig(x)` returns a matrix d whose diagonal entries are eigenvalues, and a matrix v whose columns are eigenvectors of x.

eye(n) - returns an n x n identity matrix

format long - causes numbers to be displayed in full-precision.

format short - causes numbers to be displayed with only five digits of precision.

helpwin - produces a window containing an index of help files. Double-clicking entries produces help information.

inv(x) - returns the inverse of the matrix x, if it exists.

mesh(x) - plots a mesh surface of the values in the matrix x.

norm(x) - returns the norm of the matrix x.

ones(n) - returns an n x n matrix of ones. Likewise, `ones(m,n)` returns an m x n matrix of ones.

plot(x,y) - creates a graph of y against x.

plot3(x,y,z) - creates a graph that is a projection of the segments formed by joining successive points (x_i, y_i, z_i) , where x_i is the i th element of the vector x, etc.

rank(x) - returns the rank of the matrix x.

size(x) - returns the dimension of the matrix x.

zeros(n) - returns an n x n matrix of zeros. Likewise `zeros(m,n)` returns an m x n matrix of zeros.

4.2 Elementwise Operations

We have seen that Matlab interprets all arithmetic operations in the context of matrix arithmetic. On the other hand, very often we want to create vectors of values and then manipulate the entries of those vectors individually. Matlab provides many tools to make this easy.

Suppose that we want to create a vector of integers from 1 to 20. To do so, we only need to type `variable=first:last`, where `first` is the value of the first entry in the vector, and `last` is an upper bound for the last entry. The default increment for each entry is one.

```
>> x=1:10
```

```
x =  
    1     2     3     4     5     6     7     8     9    10
```

The first entry does not have to be an integer, and the difference between the first and the last is not required to be an integer. If `last - first` is not an integer, then the largest entry in the vector is the first plus the greatest integer smaller than `last - first`. It is easier to see in an example.

```
>> x=.83:9.8
```

```
x =  
  
Columns 1 through 7  
    0.8300    1.8300    2.8300    3.8300    4.8300    5.8300    6.8300  
  
Columns 8 through 9  
    7.8300    8.8300
```

More often we want to create a sequence of points with some uniform spacing other than one. That calls for another term in the definition for the vector.

```
>> x=1:.5:10
```

```
x =  
  
Columns 1 through 7  
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000  
  
Columns 8 through 14
```

```
4.5000    5.0000    5.5000    6.0000    6.5000    7.0000    7.5000
```

```
Columns 15 through 19
```

```
8.0000    8.5000    9.0000    9.5000   10.0000
```

Note that the result was a row vector of dimension 19. Since Matlab cannot put all 19 entries on a single line, it uses two rows, and labels the columns so that we can understand what we are looking at in the vector.

Once we have a vector of values, we may manipulate it entry by entry in a variety of ways. First, we note that Matlab has many functions that apply naturally to scalars. When vectors are given as arguments to those functions, they apply elementwise. For example,

```
» x=0:.5:pi
```

```
x =
```

```
0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
```

```
» sin(x)
```

```
ans =
```

```
0    0.4794    0.8415    0.9975    0.9093    0.5985    0.1411
```

Naturally, Matlab provides all of the other trigonometric, logarithmic, and exponential functions you would expect, i.e. $\tan(x)$, $\cos(x)$, $\exp(x)$, and $\log(x)$ are all valid. On the other hand, algebraic operations are a little less transparent. For example, Matlab understands that $x^2 = x*x$. When x is a vector this is meaningless, since the row and column dimensions of x do not mesh. Thus, if we want to compute the square of each element of the vector, we need a different notation. Matlab provides this by preceding any operation you want to perform elementwise by a period ("."). In other words

```
» x.^2
```

```
ans =
```

```
0    0.2500    1.0000    2.2500    4.0000    6.2500    9.0000
```

```
» x.*x
```

```
ans =
```

```
0    0.2500    1.0000    2.2500    4.0000    6.2500    9.0000
```

In addition to operating on entire vectors or matrices elementwise, we can operate on them one element at a time. The i^{th} element of the vector x may be referred to as $x(i)$, where i varies from 1 to the dimension of the matrix. For those who are C programmers, remember that vectors in Matlab begin with the index 1. In our present example we see

```
» x(1)
```

```
ans =
```

```
0
```

```
» x(2)
```

```
ans =
```

```
0.5000
```

It is important to observe that the notation for subscripted elements is the same as that for arguments to functions. Matlab keeps track of which objects are functions and which are matrices. It is important for you to do so also - do not name your matrices `sin` or `exp`.

For matrices one uses two subscripts to identify individual elements. One may even assign values to elements in this way.

```
» B(1,1)=1; B(1,2)=2; B(2,1)=3; B(2,2)=4;
```

```
» B
```

```
B =
```

```
1    2
3    4
```

As we mentioned earlier, Matlab views vectors as $n \times 1$ or $1 \times n$ matrices. Thus, you could refer to elements of the vector x that we defined above using two subscripts.

```
» x(2)
```

```
ans =
```

```
0.5000
```

```
» x(1,2)
```

```
ans =
```

```
0.5000
```

It is also important to be able to refer to parts of matrices. In other words, we frequently need to be able to define submatrices. This is done using colons again. To refer only to the first five elements in the vector x we would type

```
» x(1:5)
```

```
ans =
```

```
0    0.5000    1.0000    1.5000    2.0000
```

Likewise to define a 3×3 matrix composed entirely of ones, we could use the colon notation on the left side of the equation.

```
» A(1:3,1:3)=1
```

```
A =
```

```
1    1    1
1    1    1
1    1    1
```

We do not have to use ranges in all indices of a matrix or vector. For example, to refer to the first column of A , we use the colon notation only for the first index of the matrix A , vis.

```
» A(1:3,1)
```

```
ans =
```

```
1
1
1
```

There is no need for the element specification to start with the first element. For example, we could define a unit vector $u = (1 \ 0 \ 0 \ 0 \ 0)$ using the commands below.

```
» u(1)=1;u(2:6)=0
```

```
u =
```

1 0 0 0 0 0

EXAMPLE

Consider the salmon data from the example chapter. We want to use Matlab and a simple least-squares process to fit a line to the control group of those data.

First, we type in the data vector.

```
» y=[4.2 8.5 8.6 9.4 9.8 10.3 11.2 15.5]';
```

Next, we need to form basis vectors for the space from which we intend to approximate the data. Since we want an approximation that is a polynomial of degree one, we shall use functions $f_1(x) = 1$ and $f_2(x) = x$ to generate vectors that we use for the approximation. We must evaluate these functions at the x-values for the data to get the vectors. It is easy to see that one vector will be composed simply of ones, while the other will have entries that are the days on which the data were taken. We use the commands below.

```
» v1(1:8)=1;
```

```
» v2=[-36 0 4 16 26 49 80 208]';
```

We make a matrix A to use in solving for the coefficients of the approximation out of the column vectors we have just defined.

```
» A=[v1 v2];
```

Since the problem is really very small, we just invert the matrix in the normal equations to solve for the coefficients. To invert matrices, Matlab provides a built-in function called `inv()`.

```
» coeffs=inv(A'*A)*A'*y
```

coeffs =

7.9732

0.0395

We can return the values of the line we have found at the x data points by multiplying the coefficients by A. This has the effect of multiplying the first column of A by the first coefficient and the second column by the second coefficient. Since the columns of A were the basis vectors for the space from which we approximated the data, the result is the least-squares fit to the data.

```
» line=A*coeffs
```

line =

6.5505

7.9732

8.1313

8.6056

9.0008

9.9098

11.1350

16.1938

4.3 Matlab Plots

Matlab provides very powerful tools for plotting functions, data, surfaces, and almost anything else you need. It can save its graphics in a variety of formats, including postscript, bitmaps, png, jpeg, and gif images. Thus, it is ideal for creating graphics for publications or for the Web. There are some caveats in order here. The student version of Matlab does not provide the ability to save plots in graphic formats - indeed, it scarcely makes provision for saving graphics at all.

The basic tool we use in creating graphics is the `plot` command. For example, suppose that we have defined `x` to be a vector of equally spaced points on the interval $[0, 2\pi]$ and `y=sin(x)`. Then we may plot those using the command `plot(x,y)`. This simple command yields the picture below.

Obviously there are many ways in which you might want to customize the appearance of this plot. For example, you should label the axes, and give it a title. That may be done in either of two ways. You might choose to use the following commands to make the labels.

```
» xlabel('x values')
» ylabel('sin(x)')
» title('Demonstration Plot')
```

Note that Matlab uses single-quotes to enclosed text strings. The quotes do not show on the graphic.

Alternatively, you could choose to apply the labels directly to the figure through the menu provided on the window containing it. This option again depends on which version of Matlab you use. If you have the student version, then the graphic appears in a window that does not allow you to do much besides print it. On the other hand, if you have the full version of Matlab, then there is a menu that allows extensive changes in labels, sizes, and other attributes of the plots.

Another way in which you need to customize the appearance of the plot is through the line or point style used for a curve. For example, very often one needs to plot data points as well as a curve that interpolates or approximates them. Many programs require you to create two separate plots to do that, and then combine them using some third command. In short, for many - perhaps most - plotting programs, it becomes a complicated operation. By contrast, for Matlab it is easy.

EXAMPLE

We created a curve to fit the data for the control group in section one of this chapter. Now we want to plot the results. We plot the data points as green diamonds, while the line that approximates them appears as in red, and is dashed.

```
» plot(x,y,'gd',x,line,'r--')
```

The plot that results appears below.

4.4 Matlab Programming

Matlab is a nice enough program for working interactively as we have described. However, the command-line interfaces might seem old-fashioned to people raised on modern GUI interfaces with their full complement of buttons, drag-and-drop tools, and menus. On the other hand, the real power of matlab lies in its programming capabilities. By now we understand that whenever we have a command line interface we also have a powerful programming environment, and this is no exception. By saving strings of commands in a text file and calling it from Matlab, we can execute complicated tasks with a single command line. We can test complex algorithms without having to type in all the steps each time we want to run it. Most importantly, we can let Matlab use its first-class linear algebra tools to do much of the detail work for us, leaving us to make algorithms using broad brush strokes.

In the simplest format, a Matlab program is just a file containing one or more matlab commands

4.5 Matlab Symbolics

5 GRAPHICS AND INTERACTION