# Introduction

We believe that the computer revolution has left most of you behind. Steve Jobs had similar thoughts when he founded Apple Computer and set out to build "computers for the rest of us." The idea was to enable people who were not computer experts—like artists, educators, and children—to take advantage of the power of computing. The graphical user interface (GUI) popularized by Apple was wildly successful, widely copied, and is now the standard interface of almost all personal computers. Thanks to this interface, people from all walks of life use computers.

Now we need to make "computers for the rest of *you*." We need computers that respond to the rest of your body and the rest of your world. GUI technology allows you to drag and drop, but it won't notice if you twist and shout. It's made it easy to open a folder and start a program, but we'd like a computer to be able to open a door or start a car. Personal computers have evolved in an office environment in which you sit on your butt, moving only your fingers, entering and receiving information censored by your conscious mind. That is not your whole life, and probably not even the best part. We need to think about computers that sense more of your body, serve you in more places, and convey physical expression in addition to information.

In more than a decade of teaching physical computing at New York University's Tisch School of the Arts, we have found people from very diverse backgrounds looking to bridge this gap between the physical and the virtual. Perhaps you are a sculptor who would like different sounds or videos to play depending on where a person touches your sculpture, or a dancer who wants a knee bend to cause bells to ring. Maybe you are a sociologist who needs to automatically log how many people pass a street corner. Maybe you're a teacher who wants to make tools for children to understand the world by doing rather than just reading. Or maybe you just want your window blinds to be lowered automatically in the afternoon if it's hot outside. Regardless of your background or technical experience, this book is designed to help you make a more interesting connection between the physical world and the computer world.

## How We See the Computer

When asked to draw a computer, most people will draw the same elements: screen, keyboard, and mouse. When we think "computer," this is the image that comes to mind. In order to fully explore the possibilities of computing, you have to get away from that stereotype of computers. You have to think about *computing* rather than *computers*. Computers should take whatever physical form suits our needs for computing. So what is computing good for?

One common reply is that computing is like human thinking. The area of Artificial Intelligence (AI), using computers to imitate, and maybe someday replace, human beings, has been an important part of computer science since its beginning. Robotics is the physical equivalent to AI. The technology you will learn in this book is very similar to what you'd learn in a book on robotics, but our typical applications are different. In robotics, people generally build robots—things that try to imitate the autonomy of human beings. We have nothing against robots, but we find the best robots much less interesting than even the dullest people (for now). Our approach comes out of a different area of computing called Intelligence Amplification (IA). This approach looks to people to supply the spark of interest and computers to capture and convey a person's expression. Rather than trying to imitate the autonomy of human beings, we want to support it. IA treats the computer as a medium of communication between people.

So what does computing offer as a medium? It can store sounds and images, but so could previous media like magnetic tape and movie film. With film and magnetic tape, information and images must be called up sequentially, according to their physical location on the tape or film as it rolls along. Ideas can only be directly linked with the previous and next idea in the sequence. Because of this, these are called *linear media*. Computers offer a break from linearity. With *random access media*, non-sequential parts of a computer's memory can be called up as if they were next to each other. This allows any idea recorded in memory to appear as if it's next to any other idea. When you combine random access with networked communication, you can display information and images stored on different continents as if they were stored next to each other. Reordering and making multiple versions are all made much easier, as anyone who has used a computer's copy and paste functions understands. Computers reduce the barriers of time and space when playing with and rearranging ideas. As a result, they better depict the changing and manifold relationships between ideas in human thought, and they can be more egalitarian in giving voice to multiple versions of those relationships.

Even if you're not out to save the world by annihilating time and space, computational media offer some concrete advantages. Without a computer, you can connect a button being pressed to a light turning on. With a computer, you can make the relationship between the button and the light more complex. For example, you can make the light's turning on dependent on the number of times the button was pressed, for how long it was pressed, or whether it was pressed in conjunction with other buttons in other rooms or on other continents. You can change the relationships on the fly; for example, you can make the light come on after two button presses during the day, and after only one button press at night. To get the computer to make these relationships between events it senses and events it causes, you write computer programs. The intelligence amplification approach counts on human beings to make the most interesting relationships, so your programs for physical computing are often relatively simple.

## How the Computer Sees Us

If you want to put the computer in a role that supports people (rather than the other way around), you need to look at the person and her environment to determine what needs to be supported. So what does a person look like to a computer? Ask this question, and you're likely to get a bunch of blank stares. Why should we care? A computer's image of human beings is reflected by its input and output devices. In the case of most desktop computers, this means a mouse, a keyboard, a monitor, and speakers. To such a computer, we might look like a hand with one finger, one eye, and two ears (see Figure I.1). To change how the computer reacts to us, we have to change how it sees us.

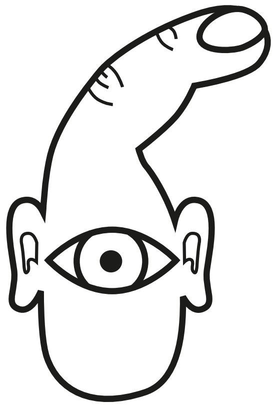


**Figure I.1**
How the computer sees us.

The human being as seen through the computer's input devices is a sad creature. Kurt Vonnegut's Tralfamadorians from *The Sirens of Titan* look much like this, and their perspective is as alien to ours as this poor creature's. It can't walk, dance, or jump; it can't sing or scream. It can't make grand sweeping gestures. And it has only one direction in which to look.

Before we invent new forms for the computer, we need to decide why it needs to take new forms. We need to take a better look at ourselves to see our full range of expression. This includes everything from the spontaneous expression on your face to the more deliberate expression of a trained artist. Just in the act of standing up, a person effortlessly reveals important details through hundreds of subtle and unconscious adjustments every second. Even though these expressions come and go very quickly, humans have amazing abilities for reading into this body language the interior state of another person. To make the computer a medium for expression, you need to describe the conversation you want to have with (or better yet, through) the computer. For example, in a Web chat room, should the *context* of the expression—that is, the posture of the user—accompany the *text* of the chat? You also need to examine your environment. Does life continue when you leave the swivel chair? Should the computer be able to interpret this action? Do people prefer to vote with their feet? How do you record their vote? Once you've taken these steps, you'll be able to realize more of the physical potential of computers, and also that of human beings.

## The Concepts

There are a few key concepts that come up repeatedly throughout this book, so it's worthwhile to introduce them briefly here. Physical computing is about creating a conversation between the physical world and the virtual world of the computer. The process of *transduction*, or the conversion of one form of energy into another, is what enables this flow. Your job is to find, and learn to use, *transducers* to convert between the physical energy appropriate for your project and the electrical energy used by the computer. To cut this task down to size, it helps first to identify the direction of the energy flows as *input* or *output*, and then treat each flow as a separate problem. You will learn that the signals in these energy flows can be viewed as *digital* or *analog*. Identifying how you want to view the flow will help both to clarify the interaction you are creating and to further narrow your search for transducers. Being able to identify how events in the flow occur over time, whether they happen *serially* or in *parallel*, will help determine how best to plan the interaction.

## Interaction: Input, Output, and Processing

When people talk about computers, they often say that computers are useful because they make things interactive. "Interactive" is a fuzzy term, and often misused for all kinds of ridiculous purposes. Author and game programmer Chris Crawford has a great definition for it: *interaction* is "an iterative process of listening, thinking, and speaking between two or more actors." Most physical computing projects (and most computer applications in general) can be broken down into these same three stages: listening, thinking, and speaking—or, in computer terms: input, processing, and output. Breaking down your project along these lines will enable you to better focus on your particular challenges and possibly to skip entire sections of this book. In Chapter 8, "Physical Interaction Design, or Techniques for Polite Conversation," we will return to this three-part cycle of events to create interactions that balance them in a satisfying way, like a good conversation.

### *Input*

For many people, input is all they want to learn from physical computing. They are already happy with their ability to express themselves on a computer, either through the screen or through the speakers, but feel constrained by the input of a mouse and keyboard. Input is usually easier than output because it takes less energy to sense activity than to move things.

### *Output*

The most provocative physical computing projects are ones that don't just sense the world; they also change it. In general, physical output can be more difficult than input because it often requires electrical (as opposed to electronic) and often mechanical skills. There are a couple of devices for light, sound, and movement that are very easy to use, which we will cover in Part I of this book. You can also get fairly far rather easily by connecting your physical input to a desktop computer, which has great capabilities for sound and video output. In Part II, we will meet the challenge of output in depth, using motors and other devices to move things in the physical world.

### *Processing*

Input and output are the physical parts of physical computing. The third part requires a computer to read the input, make decisions based on the changes it reads, and activate outputs or send messages to other computers. This is where programming comes in.

## Transduction

One of the main principles behind physical computing is *transduction*, or the conversion of one form of energy into another. A microphone is a classic transducer because it changes sound pressure waves in the air to a changing electrical voltage. Speakers convert the same energy in the opposite direction. Transducers are the eyes, ears, hands, legs, and mouth of any physical computing system.

Much of the challenge of physical computing is converting various forms of energy, such as light, heat, or pressure, into the electronic energy that a computer can understand. Sometimes it's easy to find the right transducer for the job; at other times, you will contrive the interaction to fit a transducer that you know how to use.

Input transducers (sensors), such as switches and variable resistors, convert heat, light, motion, and sound into electrical energy. Output transducers (actuators), such as motors and buzzers, convert electrical energy into the various forms of energy that the body can sense.

## *Digital and Analog*

When describing an activity, begin by breaking it down in terms of how many possible outcomes there are. Sometimes we view events in the world along a continuous range of possible states. At other times, we only care about the difference between two possible states. When two states will suffice, we'll call it *digital*. When a continuous range of multiple states is considered, we'll call it *analog*. For example, as you get dressed in the morning you might prefer to know the actual outdoor temperature (analog) rather than just hearing that it's hot or cold (digital).[1] On the other hand, when deciding to bring your umbrella, you only want to know whether it is raining or not (digital); you don't care how hard it's raining (analog). In general, digital input and output (I/O) are easier than analog I/O because computers use a two-state, or binary system, but analog I/O can be more fun and interesting.

The language you use to describe the project will tip you off to whether your I/O requirements are analog or digital. For example, if you can use the words "whether or not," or the word "either," in describing the input or output, then you're probably talking about a digital input or output. If you can use words like "how much" for input or superlative adjectives like "stronger," "faster," "brighter," then you're probably talking about an analog input or output. For example, a digital output would work to *either* turn a light on or off; an analog output would be required to determine whether the light is *brighter* or *dimmer*.

## *Parallel and Serial*

The terms digital and analog make it possible for us to be clear about what we're listening to (our input) or what we're saying (our output). We also need to be clear about how we're speaking or listening. Sometimes we present ideas simply, one after another, in discrete chunks. For example, a simple melody played on a solo instrument lets us focus on the structure of the melody, and how its changes affect our emotions. At other times, we present many ideas all at once so that they complement each other. For example, a symphony's power comes from the experience of hearing many instruments playing different harmonies all at once; each individual instrument's melody line is important, but the combined effect of all of them presented at once is what we take away from the experience.

To describe the order in which events happen, we can talk about them happening either one after another in time or all at once, simultaneously. For our purposes, we'll refer

[1] The truth is that analog and digital may not be the most accurate terms. Terms like multi-state versus two-state or continuous versus binary might be better. But digital and analog are commonly used terms among the manufacturers of the tools we will be using.

to events that happen one at a time as *serial* events, and when several events happen simultaneously, we'll refer to them as *parallel* events.

While we're using these terms in a broad sense, to talk about how events are organized in time, we'll also use them to refer to more technical aspects of the work as well. You'll see how electrical energy can flow through components serially (one after another) or in parallel (through several components at the same time), and we'll talk about how computers can exchange bits of information serially or in parallel as well.

## The Practice

Physical computing is best understood by doing it rather than talking about it, so in this book we focus primarily on how to do it. Following are a few general guidelines that will help you keep your wits about you in the midst of all the technical information that follows later in this book. If you find yourself getting lost in the details, come back to this section and use it as a guide to regain an overview of your whole project.

### Getting Started: Describing What Happens

The first step in a physical computing project is to describe what you want to happen. If you can't first describe what happens in plain language, it will be difficult to write the programs and build the circuits to make it happen. Describe the whole environment of the project from the point of view of the person experiencing what you're making. Describe what she sees, hears, and feels and what she can do to change the environment. Describe the experience as it unfolds, what changes as the person takes various actions, and how her attention and actions are focused by the changes. Describe why this is engaging to the person and how the sequence of events should work to keep her engaged. You'll revise this description several times as you realize the project, so don't worry if some details are missing. On the other hand, don't let the process of implementation distract you from filling in the missing details as you go.

Focus your description on *what* happens, not *how* it happens. Avoid describing the specific technologies involved or the tools used to make things happen. These details will prejudice your thinking and possibly cripple your concept. Frequently, we've had students skip to the technology, coming to ask how to implement some esoteric and difficult-to-use sensor. Our first question is always, "What are you using this to do?" Quite often, once they describe what they want to happen without describing the technology, a simpler solution can be found.

For example, say you want to announce guests at a party in a big way. When a person walks into the room, a theatrical curtain opens, a bright spotlight hits the person, and loud applause is heard. This description tells you nothing about the technologies that make it work, but it gives you enough description to start to plan how to make it a reality. You know you need a curtain, a spotlight, and applause, and you know you need to be able to sense when a person enters the room.

After you've described the project and iterated your concept a few times in plain language, without thinking about the technology, you should break the project down into the stages of input, output, and processing. For example, the input in the example above would be the person walking into the room, the output would be the spotlight and the applause, and the processing would be turning on the light and playing the applause *if* a person walks in.

Next, identify your input and output as digital or analog and begin your search for the perfect transducers. Again, in the example above, if you wanted the volume of the applause to depend on *how far* the person walked into the room, you would need an analog input and output. If you wanted the applause to either be on or off, depending on whether or not the person was in the room, you would need digital input and output. It will help you to focus in on the most relevant parts of this book if you can break your project description into parts that fit into the categories shown in Figure I.2. Use this or a similar worksheet to fill in the input/output needs of your project.

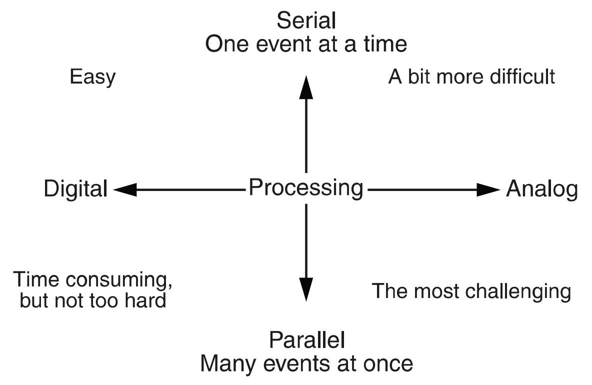
**Figure I.2**
Categorize your physical computing challenges.

| Digital Input | Analog Input | Processing | Digital Output | Analog Output |
|---|---|---|---|---|
| | | | | |

In addition, you should describe the sequence of events. Does the light happen before the applause? Or do they happen at the same time? In the former case, they'd be *serial* events, and in the latter, they'd be *parallel* events.

Refer to the chart in Figure I.3 to help figure out how complex your project is, and what needs to be done.

**Figure I.3**
Mapping your project: analog and digital, serial and parallel.



## Level of Abstraction (and Distraction)

With any technical practice, you inevitably have to make strategic decisions about the level of abstraction between you and your tools. Higher-level tools place you at a higher level of abstraction from the details of the technology.[2] As a result, they are easier to use but don't

[2] This way of thinking of high levels and low levels may seem counterintuitive if you're used to thinking of "higher level" meaning more advanced technologically. Instead, think of "lower level" meaning a lower level of padding between you and the metal of the computer. We think a little padding goes a long way.

always allow you to do everything you would like. Our approach starts at the highest level that still gets the job done and works down when necessary. With high-level tools, you can quickly try a new idea, and if it doesn't work, you can move on before you get too invested technically and emotionally. In technology, tools change rapidly enough that a high-level approach works in your favor: tomorrow's high-level tool will have the power of today's low-level tool.

In practice, though, it's never that clear. There are temptations in lower-level tools to lead you astray. For example, if you are a food lover, you might be attracted to cooking from scratch, regardless of whether it tastes better, because you enjoy the process. Be aware that you may be indulging a technical machismo that will be distracting, time-consuming, and will probably yield a less impressive result. Just because you made your crème brûlée from scratch doesn't mean your guests are going to like it (especially if you've never cooked it before). On the other hand, when you know something about cooking, it's difficult to make a signature dish using only pre-prepared foods. If you are attempting something very specific and unusual, there will come a time when it's easier to do it yourself than to find, cobble together, and then work around a bunch of mix-and-match prepared solutions. A combination of working at the highest level, knowing what's available at lower levels, and knowing when to switch up or down, will yield the best results (see Table I.1).

**Table I.1**
Levels of Abstraction

| SOFTWARE | FOOD | MICROCONTROLLERS |
|---|---|---|
| Higher Level ("Hello World!") | Higher Level ("Hello, may I take your order?") | Higher Level ("Hello World!") |
| MAX | Ordering out | Teleo |
| LINGO/ACTIONSCRIPT | TV dinners | BASIC STAMP 2 |
| PROCESSING | Hamburger Helper | BX-24 |
| JAVA | Using the deli counter at the supermarket. | Basic Atom Pro24 |
| C | Using produce and the butcher at the supermarket. | PIC |
| ASSEMBLY | Growing your own foods, harvesting them, and preparing them from scratch. | SX |
| Lower Level ("1001001 0110110") | Lower Level ("Henry, go kill me a chicken, and we'll have some pot pie tonight.") | Lower Level ("1001001 0110110") |

## The Tools

We will give examples at different levels, but our inclination will be toward tools in the middle to high level. To make the connection between the physical world and the digital, you'll learn to assemble circuits, connect them to computers, write software for the computers, and enable computers to communicate with each other (see Figure I.4).

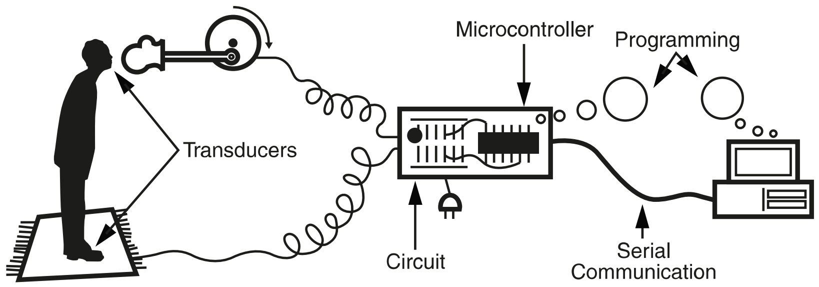


**Figure I.4**
The parts of a physical computing system.

## Circuits

You will have to build a little circuitry as the glue between the transducers you use to sense and control the world and the computers you use to interpret what's going on. For the majority of common transducers, you will copy one of four or five basic circuits we'll lay out in the early chapters. Building these circuits is fairly simple. It amounts to connecting a few wires and an electronic component or two.

While it will help to have some feeling for how electricity behaves, we're aiming to make you do the least amount of work to get information from the physical world through sensors into the computer. We'll cover the basics needed to understand the circuits we're using, and point to other sources for more detail. In a sense, the computer is the mother of all general circuits, and you can finesse the connection between input and output further in software. You can get far in physical computing with the most basic understanding of electricity.

Circuits are usually described in a diagram called a *schematic* that shows the electrical components and how they are connected to each other. You will need to know enough about schematics to be able to read them, but to get started you need not be able to draw schematics or design circuits.

As you get more adventurous with your transducers, the translations of energy will get a little more involved. Then you will need to learn more about the behavior of electricity and how to build circuits, particularly when dealing with more powerful output devices like motors.

## Computers

The word "computing" might seem at odds with the word "physical." One of the main strengths of computer technology is transcending the time and space of the physical world. Yet physical computing is all about recognizing that people are still 99 percent monkeys who really enjoy the pleasures and constraints of the physical world. In physical computing, we want it both ways: we want the liberation that computers allow situated in the sensual world that humans enjoy. To do this, we'll use a variety of computers, but always do our best to put them in the background so that we can focus on the experience between humans in the foreground.

### *Microcontrollers*

The main computer we'll use in physical computing is the microcontroller. This is a very small, very simple computer that's good at three things: receiving information from sensors, controlling basic motors and other devices that create physical change, and sending information to computers and other devices. They act as gateways between the physical world and the computing world. Microcontrollers are often at the heart of complex electronic devices, so understanding how they work will give you new insight into electronic devices that you already own

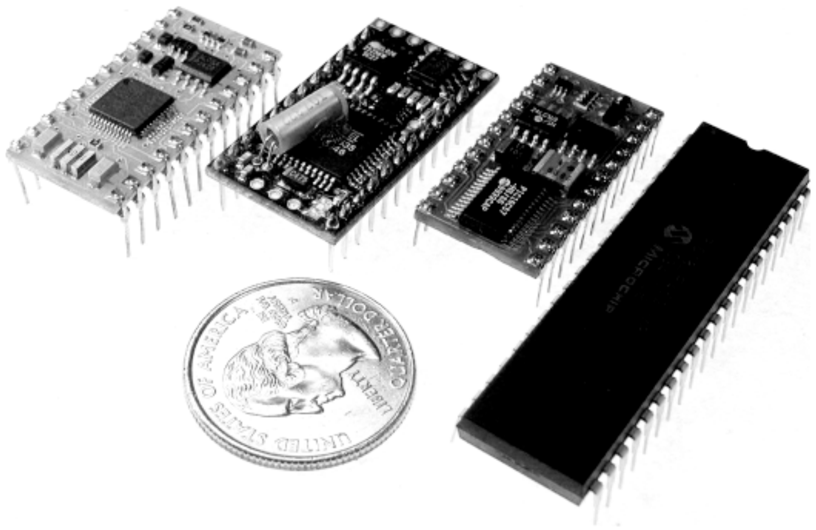


**Figure I.5**
The four microcontrollers discussed in this book. These simple computers are at the heart of many physical computing applications.

Microcontrollers are small and cheap. This allows you to explore location-specific projects that embed computers in the most unlikely places, like shrubbery and sneakers, or to develop projects where the actions of many simple devices add up to a more interesting whole.

Microcontrollers are found in everything from washing machines to light switches. You benefit from this ubiquity, as it has brought down the cost and improved the ease of use of microcontrollers.

## *Multimedia Computers*

To some degree multimedia computers (desktop and laptop computers) are what we are working against in physical computing. These computers presume that the person using them will be relatively inactive, except for her fingers and hands, and that her eyes and ears will be focused in one direction. These computers may be multimedia-capable on the output side, but they are not so on the input side. One of our main objectives is to get people to picture a computer as something other than a couple of big beige plastic boxes on a desk and to picture their interaction with computers as something other than typing and clicking. The problem with our zeal to stretch your concept of computers beyond multimedia computers is that they are so useful, particularly for tasks such as generating sounds and graphics and sensing physical activity through cameras and microphones. Many projects combine the interesting input and output possibilities of microcontrollers with the multimedia output capability of multimedia computers. On the other hand, if your project does not involve any multimedia, such as playing sounds or videos, you may not need the complication and expense of a multimedia computer at all. Connecting back to multimedia computers is one of the things that separates this book from books about robotics. Robotics books tend to insist on having the microcontroller stand alone. We're not so swift to dismiss the multimedia computer's output capabilities when it's useful for communicating with people or between people. Multimedia computers are also useful for prototyping part of a project that ideally will be small and portable, but is not easy to miniaturize quickly. For the purposes of a demonstration of a futuristic project you might prefer to just say "Pay no attention to that huge computer behind the curtain."

### *Intermediate Computers*

There is a wide range of computers between multimedia computers and microcontrollers. For example, notebook computers, tablet computers, single-board computers, palmtop computers, and mobile phones are all types of computers that might fit perfectly into your particular physical setup. Most of these intermediate platforms use operating systems and development environments that resemble the multimedia computer's, so our material on making a connection between the microcontroller and the multimedia computers will give you a leg up with these platforms.

## Programming

This will send many readers running for the doors because they've tried and failed in the past to learn programming. In fact, physical computing is an excellent environment to learn computer programming. Abstract programming concepts like bits and bytes are embodied by tangible things like switches. In addition, the programs for microcontrollers tend to be very small and simple. There are only a few things you might want to do on a microcontroller: read sensors, turn things on or off, and send messages to other computers. Often it only takes a few lines of code, and much of that code can be borrowed from others and modified to suit your purposes.

You have a choice of many languages and microcontrollers, but we will be giving our examples for programming microcontrollers in one of the friendliest languages, BASIC. The process of programming microcontrollers involves typing out the programs on a multimedia computer and downloading them into the microcontroller. Chapter 5, "Programming," is geared toward someone who has little programming experience. If you are an experienced programmer, you can probably just skim the examples to get the syntax.

Programming multimedia computers, on the other hand, is a big subject. The topic of programming is too broad to be covered in one book, so our focus will be on how to get computers to communicate basic information with each other. If you already have some experience programming in Director/Lingo, Max/MSP, Processing, or Java, you are in perfect shape for this book because we will show you how to communicate between the microcontroller and the multimedia computer in these languages. Beyond communicating with the microcontroller, programming multimedia computers for the multimedia needs of your project is too idiosyncratic for us to cover properly here. If you are new to programming in general, you will need to pick a multimedia programming environment and learn it. We recommend those mentioned above, and we will provide a few examples using them to get information from a microcontroller into a multimedia computer.

## Communicating between Computers

We rarely talk about computers anymore without talking about a network of computers. Even if you are not sending messages across the Internet, you might need to communicate between two different types of local computers. For example, your microcontroller is good at listening to switches, but not so good at more advanced multimedia tasks. It might send messages to your multimedia computer, which is better at playing sounds or videos. There are many different ways to communicate between computers. We'll be introducing a method called *serial communication* that offers the most flexibility for the least amount of work. We will also talk about more specialized versions of this method, such as MIDI and Internet protocols.

## Your Concept: Don't Lose It

This book is about working backward from your project idea to the specific techniques you need to know to realize it. The journey from the concept of the project to realization is seldom one-way. The technical skills you develop along the way will inform and change the concept. After you develop some fluency with the tools, ideas often come concurrently with the making of the project, not necessarily before. But if this is your first experience with these technologies, it's easy to lose your way.

There are two big traps along the journey into physical computing. The first and more pleasant of the two traps is technological seduction. It's possible to get so pleased with your new technical powers that you dig into unnecessary technical detail or start growing weird new limbs for your project. In practice it's hard to tell the difference between when technical obsession will result in a very subtle and unexpected project and when it will just lead to lonely mutterings to yourself. It's a good idea to check your work with a potential audience as you go. If your audience doesn't notice any improvement in a project as a result of a particular technical change, you might want to re-evaluate how necessary the change is.

The second trap is spinning your wheels for so long, trying to get something to work, that you give up on the entire project in frustration over one part of it. Here again, sometimes sidestepping a technical problem will require ingenuity that may totally jumpstart and liberate your project; other times it will leave a glaring compromise in the final product.

There are four things that can keep you focused as you implement your ideas. First, keep a journal of the journey. Write down your ideas as you go, as well as the questions you have, the problems you encounter, and the solutions you come up with. This helps you to remember where you were going before you got discouraged by a technical or conceptual problem. In fact, your best entry may be the one you make right at this moment, recording what got you going down this road before you lost your technical innocence (assuming you had any to begin with). A healthy process is one in which you take frequent breaks from the details of realization to look at the overall idea, so don't wait until you're discouraged to revisit your journal. Better yet, make it a public Web log so other people can benefit from your progress.

Second, work fast and at a high level. Whenever possible use prefabricated technical solutions to at least test things. Don't spend your time perfecting endless details until you have proven the overall concept. The longer you spend implementing something, the more invested you will become in it and the less objective you become about its actual value to the project.

Third, don't become paralyzed by planning. Unless you're psychic, it's better to just try something and see how it works out. If the first solution doesn't work, try another. Each variation will give you new ideas on what's good about your project and what's not.

Furthermore, being less invested in any one solution at the beginning will make it easier to find a workaround or a different solution when you hit an obstacle later in the process.

Fourth, collaborate with other people. Explaining yourself, particularly to people who do not think like you, will keep you honest. When you have checked everything a hundred times and still can't imagine what could be causing the problem, a fresh set of eyes is the best solution.

Finally, take frequent showers and work on many parts of the projects at once. A lot of solutions will appear in your peripheral vision, so taking frequent breaks or switching tasks will help.