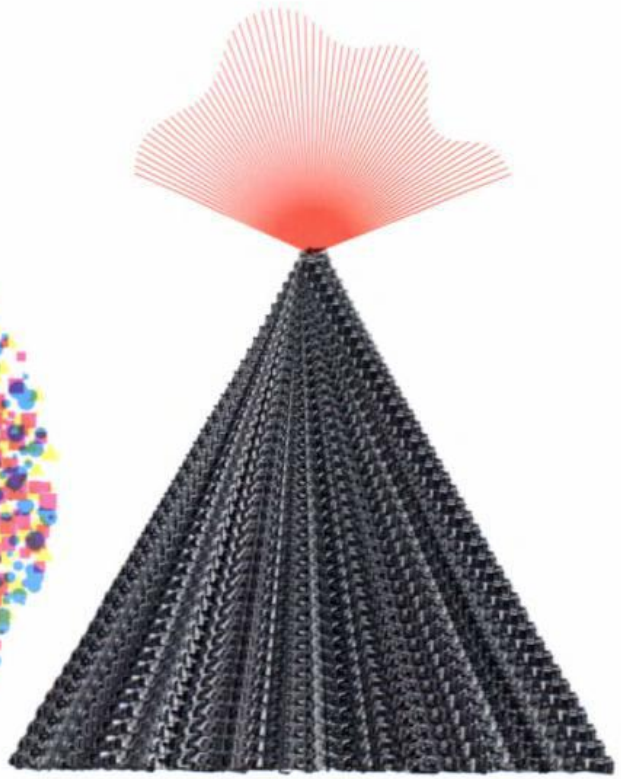
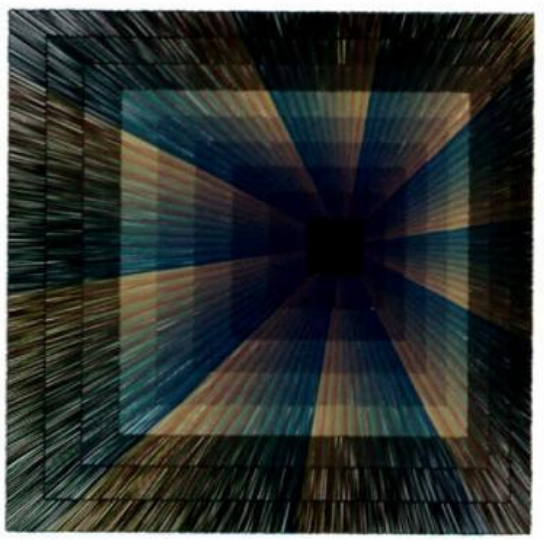
John Maeda

Design By Numbers

foreword by Paola Antonelli





First MIT Press paperback edition, 2001

Designed by John Maeda.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Maeda, John.
Design By Numbers / John Maeda
p. cm.
Includes bibliographical references.
ISBN 0-262-13354-7 (hc. : alk. paper), 0-262-63244-6 (pb)
1. Computer programming. 2. Computer Graphics. I. Title.
QA76.6.M336 1999
005.1—dc21 98-37583
CIP

Paola Antonelli **Foreword**

About fifteen years ago, the desktop computer entered the professional world. This fantastic brand-new medium demanded and deserved experimentation. Wherever it was implemented in the world of the arts, it changed not only the final product, but also the creative process itself. As always happens with disruptive innovations, the computer provoked much criticism. Self-discipline, its detractors said, was being abandoned in favor of less rigorous trial-and-error, cut-and-paste habits. The attraction of immediate results appeared to work against the strategic thought made necessary by earlier technologies. Thanks to the computer, writers could jot down concepts, and designers visual fragments, apparently at random and without a strong leading idea, rearranging them only at the end to form a finished text or design. Because of this apparently time-saving technology, its detractors concluded, texts, images, and objects would lose power and generate a world of diluted intensity and rigor.

In graphic design, in particular, the computer created a tidal wave whose impact was quick and massive. It quickly generated factions and ignited debate. On one side, against the computer, sat some of the best champions of hand-made modernism, like Paul Rand and Massimo Vignelli. On the other side sat many young talented designers and thinkers, like the group Emigre and Lorraine Wild, who were confident enough in their self-discipline and ability to exploit and ride the new medium. Modern design is about showing clarity of process and purpose, and the best among them relied on their post-modern flexibility to update the positive qualities of modern design and to express the most contemporary visual culture. As a matter of fact, modern design objects display in their function and form the process that generated them. Such objects are determined not by the process, but rather by the use that is made of it. When this characteristic is taken as a definition, modern design becomes timeless and styleless and can be found alike in ancient Greece, in 1938, and today.

The battle that is now history had many famous episodes and many less famous epilogues. Among the most emblematic is the encounter between Paul Rand, the idolized and cantankerous master who believed he rejected the new, and John Maeda, a younger designer who could play his computer like a violin, who happened to be a fervent fan of Rand. Maeda showed Rand that the computer is indeed a powerful tool that can be used to produce powerful designs, by generating with it brand new examples of historical modernism.

The most important part of Maeda's production, and the one he is most proud of, is not the final object, but rather the process. In his work, the process is the core that informs the final outcome. Maeda's fundamental idea is that to successfully design with a computer, one has to design, or at least understand, the program one uses. This position brings him close to post-modern pioneers like Rudy VanderLans and Zuzana Licko of Emigre, who in 1985

designed on their small Macintosh the low-resolution typefaces that they used in their graphics. It also brings Maeda close to the best contemporary industrial designers, who not only master the design process, but also influence the design of the materials they will use.

By stressing the necessity of knowing in depth the tools and methods of design, Maeda also responds to one of the major criticisms advanced by the detractors of the computer, the lack of self-discipline that it allegedly allows. Even though it is true that, in general, the new fields of design, like computer-based graphic design ten years ago and websites or wearable computing today, seem to attract many amateurs and to provoke in designers a short circuit that erases the school memory of a correct design process, it is also true that it always takes time. Graphic design is undergoing a postindustrial revolution similar to the industrial revolution that shook object design more than a century ago. It took some time and some polemic before design thinkers could give order to the exuberance of the hideous new objects, such as beflowered cast-iron toilets and chairs, that sang the praise of the industrial technology in the second half of the nineteenth century. It is still taking time for the graphic design profession as a whole to feel comfortable with this new and ever-improving technology.

With this book, Maeda teaches both professional and amateur designers a design process that paradoxically has a hands-on, almost Arts and Crafts feeling. His approach to computer graphic design is not different from an approach to wood carving. Aimed at exemplifying the basic design of a programming method that transforms itself into a visual design process, this book is an unmatched attempt to share precious knowledge. As awareness of the new medium continues to disseminate, the more designers will learn to incorporate the digital medium in their practice without being unsettled by it.

Paola Antonelli
Associate Curator
The Museum of Modern Art

```
/*** DBN VERSION 0, by John Maeda. An attempt to create a programming language that can teach the basics of computation to the non-mathematically inclined person. ***/ import java.awt.
static int tMSG=0, tCMD=1; Image ic[]; Image ic2[]; String dir = "stuff/"; Image title; Image imp, ims; bufpanel bp; env app; Rectangle loadr = new Rectangle(); Rectangle saver = new Rectangle
app.getImage(app.getDocumentBase(),dir+"play.GIF"); for(int i=0;i<=100;i+=10) { int v = 100-134*i*i/10000+34*i*i*i/1000000; System.out.println(i+": "+v); } } Graphics myg=null; int pw; public
setmsg(String s) { msg = s; repaint(); } public Dimension preferredSize() { return new Dimension(100,24); } public boolean mouseUp(Event ev, int x, int y) { if (x<24) { if (!playp) { app.goaheadand
app.doload(); } else if (saver.inside(x,y)) { // System.out.println("saver....."); app.dosave(); } return true; } Font f = new Font("Helvetica",Font.PLAIN,10); public void paint(Graphics g) { Rectangle
24,0,24,r.height); g.setColor(Color.darkGray); g.fillRect(0,0,r.width,r.height); g.setColor(Color.white); g.setFont(f); pw = g.getFontMetrics().stringWidth(msg); g.drawString(msg,23,15); g.setColor(
g.fillRect(saver.x+4,saver.y+4,saver.width-8,saver.height-8); g.setColor(Color.white); g.drawString("rd",loadr.x+4+3,loadr.y+16); g.drawString("wr",saver.x+4+3,saver.y+16); } } } class StringToke
return s.substring(start,i); } else { b=i+1; return s.substring(start,i+1); } } public String getuntilmatched(int start, char c1, char c2) { int i; int level = 0; for(i=start;i<len;i++) { if (s.charAt(i)==c1) lev
for(i=b;i<len;i++) { char c = s.charAt(i); switch(c) { case '<': return getuntilmatched(i,'<','>'); case '(': // do balanced version return getuntilmatched(i,'(',')'); case '[': return getuntilmatched(i,'
netinitp = false; boolean setting = false; int numgets; // hiresbuf int cbw, cbh; int pgcol = 0; int pencol = 100; int bw,bh; int [][]bits; byte [][]dbits; accessorypanel msg=null; long curt; Image bim
boolean stepp = false; // Image bimcp; public void createpage(int w, int h) { bw = w+1; bh = h+1; bits = new int[bw][bh]; dbits = new byte[bw][bh]; if (bim!=null) bim = null; bim = createImage(bw
= new Color[101]; for(i =0;i<101;i++) { int g = (int)((float)i*255f/100f+.5f); bitscol[i] = new Color(g,g,g); } } for(i=0;i<26;i++) { keys[i] = false; keyt[i] = -1; } } int szw = 358, szh = 318; Font f = new
bufpanel(String code, int sc) { doinits(); progstr = code; szw = 102; szh = 102; margx = margy = 4; runtimep = true; } int margx = 12, margy = 12; Image pat=null; Rectangle framer = new Recta
g.setColor(Color.white); g.fillRect(0,0,r.width,r.height); g.setColor(Color.black); g.drawRect(framer.x-1,framer.y-1,framer.width+1,framer.height+1); return; } if (pat == null) { pat = createImage(16,
g.drawImage(pat,i*16,j*16,this); g.setColor(Color.white); g.fillRect(framer.x-1,framer.y-1,framer.width+1,framer.height+1); g.setColor(Color.black); g.drawRect(framer.x-1,framer.y-1,framer.width
step; if (sc==1) step = 50; else step = 10; for(i=0;i<=100;i+=step) { int x,y; y = framer.y+framer.height-framer.height*i/100-1; g.drawLine(framer.x-2,y,framer.x-4,y); g.drawString(""+i,framer.x-4-g.
g.drawString(""+i,framer.x+framer.width*i/100+1,framer.y+framer.height+12); } g.setColor(Color.darkGray); // g.drawLine(margx,0,margx,r.height); // g.drawLine(0,r.height-margy,r.width,r.height-
gpage(int val) { if (val>100) val = 100; if (val<0) val = 0; pgcol = 100-val; clearbits(); } public void gpen(int val) { pencol = 100-val;//100-val; } public void refreshbits() { int i,j; for(i=0;i<bw;i++) for(j
public void circpts(int x, int y, int x0, int y0, int v) { setbit(x+x0,y+y0,v); setbit(x+x0,-y+y0,v); setbit(-x+x0,y+y0,v); setbit(-x+x0,-y+y0,v); setbit(y+x0,x+y0,v); setbit(y+x0,-x+y0,v); setbit(-y+x0,x+y(
r; int de = 3; int dse = -2*r+5; circpts(x,y,x0,y0,pencol); while(y>x) { if (d<0) { d+= de; de+=2; dse+=2; } else { d+= dse; de+=2; dse+=4; y—; } x++; circpts(x,y,x0,y0,pencol); } } public void grec
BresLine(x0,y0,x1,y1,pencol); if (true) return; if (dx == 0 && dy == 0) { setbit(x0,y0,pencol); } else if (dx == 0) { if (dy<0) { int dum = y0; y0 = y1; y1 = dum; } for(i=y0; i<y1;i++) setbit(x0,i,pencol);
bresen2(x0,y0,x1,y1,pencol); } else { if (dx>=-dy) bresen1(x0,y0,x1,y1,pencol); else { //System.out.println("yell"); bresen2(x0,y0,x1,y1,pencol); } }*/ } public void bresen1(int x0, int y0, int x1,
{ d+=incrNE; x++; y+=sny; } setbit(x,y,val); } } public void clearbits() { int i,j; for(i=0;i<bw;i++) for(j=0;j<bh;j++) { bits[i][j] = pgcol; dbits[i][j] = 1; } bimg.setColor(bitscol[pgcol]); bimg.fillRect(0,0,
int g = n; if (g<0) g = 0; if (g>100) g = 100; // System.out.println("g is "+g+" n is "+n); bimg.setColor(bitscol[g]); bimg.drawLine(i,j-1,i,j-1); /* myg = this.getGraphics(); myg.setColor(bitscol[g]); m
tx, ty; if (refreshp) { drawframe(g); refreshp = false; } // System.out.println("predrawbits"); // if (!bdirtyp) return; g.translate(framer.x,framer.y);//tx=(margx+1),ty=(cb.height-margy-sc*bh)); orgx = f
def_command(String s, Block b) { int i; Vector v = b.args; System.out.println("def " + s); cmdenv.put(s,b); } public void def_io(String s, Block b) { int i; Vector v = b.args; ioenv.put(s,b); } public B
System.out.println("set "+s+" to "+n); if (s.startsWith("[")) { StringTokenizer2 st = new StringTokenizer2(trim2(s)); int x = evalarg(st.nextToken()); int y = evalarg(st.nextToken()); setbit(x,y,100-n
String str = new String(st.nextToken() + " " + n); sendtoServer(str); } else { } } else curFrame().put(s,new Integer(n)); } Vector vframes = new Vector(); public void PushFrame() { // System.out.pri
return (Hashtable)vframes.elementAt(vframes.size()-1); } public int getvar(String s) throws lerr { { // need to search through all frames int i; Integer m = null; for(i=vframes.size()-1;i>=0;i—) { Hash
char word[] = s.toCharArray(); for (int i = 0; i < s.length(); i++) { if (word[i] == c) { sp("CONTAINS: "+s.length()+" "+i); return i; } } return -1; } public String comesafter(String s, int n) { return s.su
stack[] = new int[100]; int outc = 0; public int outputpop() throws lerr { if (outc == 0) throw new lerr("out stack overflow"); outc—; return outstack[outc]; } public void outputpush(int v) { // Syst
//System.out.println("new client"); client = new ClientConnection(hoster); netinitp = true; } public void settoServer(int place, int value) { //System.out.println("set " + place + " " + value); client.
//System.out.println("gotfromserver " + line + " " + numgets + "times"); return Integer.parseInt(line); } catch (IOException e) { System.out.println(e.toString()); } return -1; } public int evalarg(St
*/" s=s.trim(); // System.out.println("EVAL ARG: >>"+s+"<<"); if (s.startsWith("(")) { // find the matching paren return evalarg(trim2(s)); } // does not work, needs to seek paren at this level and d
v= 100-bits[x][y]; return v; } */else if (contains(s,'(')>0) { // got a paren up to bat // find out operator that occurs before int i; for(i=0;i<s.length();i++) { if (s.charAt(i)=='(') break; } i—; switch(s.char
evalarg(s.substring(i+1,s.length())); case '/': return evalarg(s.substring(0,i))/evalarg(s.substring(i+1,s.length())); default: System.out.println("math: unknown operator"); return 0; } } else if (cont
StringTokenizer(s, "-"); return evalarg(st.nextToken())-evalarg(comesafter(s, contains(s, '-')+1)); } else if (contains(s, '*') > 0) { StringTokenizer st = new StringTokenizer(s, "*"); return evalarg(st.n
(s.startsWith("[")) { int v; StringTokenizer2 st = new StringTokenizer2(trim2(s)); int x = evalarg(st.nextToken()), y = evalarg(st.nextToken()); v= 100-bits[x][y]; p(x+"/"+y+": "+bits[x][y]); return v;
num = evalarg(st.nextToken()); switch(num) { case 1: return mx; case 2: return my; case 3: return mb; default: throw new lerr("Unknown <loc>: "+num); } } else if (intype.equals("key")) { i
Date d = new Date(); switch(num) { case 1: return d.getHours(); case 2: return d.getMinutes(); case 3: return d.getSeconds(); case 4: return (int)(System.currentTimeMillis())/10; // break; def
((gb = get_io(intype)) == null) throw new lerr("Unknown input type: '"+s+"'"); else { Vector v = gb.args; if (v == null) { // no args doBlock(g,gb); return outputpop(); } else { // this should be
Integer.parseInt(s); } else { return getvar(s); } } // setvar A,2 // drawline A,A,10,20 final int mxstk = 10; Vector blockstk[] = new Vector[mxstk]; // public void pushblock(Vector Vector vect = null; p
msgbox("["+mx+"/" "]"); msgalive(); servicekeys(); // cmd = cmd.toLowerCase(); try { // System.out.println("doline: "+b.linenum); // while(!stepp); if (cmd.equals
"line "+(20)+" "+(20+i)+" "+60+" "+(20+i)); } System.out.println("pen 25"); {
")) { } else if {
Copyrighted material
(cmd.equals("pause")) { wt= // in 100ths of seconds t = S
} else if (cmd.equals("set")) { setvar(st2.nextToken(), evalarg(st2
is: + Block bb = get_command(cmd); if (bb == null) throw new cmd: "+cmd+" at line "+(b.linenum+1)); } else { // System.out.println("Using na
PopFrame(); } } } catch (lerr e) { if (e.lnum == -1) { e.lnum = b.linenum; throw e; } } stepp = false; drawbits(g); return false; } public void doStep(Graphics g, String a, int n, int t, int inc, Block b
doBlock(Graphics g, Block b) throws lerr { int i, j; // System.out.println("Doing block: "+b.getstr()); // b.pp(); msgalive(); //** SHOULD CHECK NARGS!!! if (b.type == Block.STR) { return doLine(g,
System.out.println("Doing a stepone: "); String line = a.s; StringTokenizer2 st = new StringTokenizer2(line); st.nextToken(); String arg = st.nextToken(); //A int inc = 1; Block nextb = (Block)v
(Block)v.elementAt(i+1); i++; Graphics g2 = bufim.getGraphics(); drawframe(g2); for(;;) { // in a forever, should do automatic double buffering // but i think this would not work if we had nest
(doBlock(g2,nextb)) { // this dont work right because doblock has to be rewriten System.out.println("ESCAPED"); break; } g.drawImage(bufim,0,0,this); } } else if (a.isa("same?")) { String li
evalarg(arg2)) doBlock(g,nextb); } else if (a.isa("notsame?")) { String line = a.s; StringTokenizer2 st = new StringTokenizer2(line); st.nextToken(); String arg1 = st.nextToken(); String arg2 = s
st.nextToken(); String arg1 = st.nextToken(); String arg2 = st.nextToken(); String arg3 = st.nextToken(); Block nextb = (Block)v.elementAt(i+1); i++; if (Math.abs(evalarg(arg1)-evalarg(arg2))<ev
nextb = (Block)v.elementAt(i+1); i++; if (evalarg(arg1) < evalarg(arg2)) doBlock(g,nextb); } else if (a.isa("notsmaller?")) { String line = a.s; StringTokenizer2 st = new StringTokenizer2(line); st.
= a.s; int nargs = cntargs(line); StringTokenizer2 st = new StringTokenizer2(line); st.nextToken(); String arg = st.nextToken(); Block nextb = (Block)v.elementAt(i+1); i++; if (nargs>2) { Vector
cntargs(line); StringTokenizer2 st = new StringTokenizer2(line); st.nextToken(); String arg = st.nextToken(); Block nextb = (Block)v.elementAt(i+1); i++; if (nargs>2) { Vector nv = new Vector();
cntargs(String s) { int i, len = s.length(); int cnt = 0; boolean lst = true, st; for(i=0; i<len;i++) { st = s.charAt(i)!=' '; if (lst!=st && st) cnt++; lst = st; } return cnt+1; } Thread runner=null; public voi
(mousex-orgx)/sc; my = (orgy-mousey)/sc; mb = mouseb?100:0; //p(mx+"/"+my); } int mousex, mousey; boolean mouseb; public boolean mouseEnter(Event ev, int x, int y) { mousex = x; mousey =
y) { mousex = x; mousey = y;mouseb = true; if (zoomr.inside(x,y)) {sc++; if (sc == 4) sc = 1; refreshbp = true; if (runner==null) repaint();} calcmouse(); if (runtimep&&((ev.modifiers)&(Event.SHIFT_M
false; calcmouse(); return true; } public boolean mouseDrag(Event ev, int x, int y) { mousex = x; mousey = y;mouseb = true; calcmouse(); return true; } public boolean mouseMove(Event ev, int x,
msg.pulse(); } public void msgbox(lerr e) { System.out.println("***MSG: "+e.getMessage()+" line: "+e.lnum); if (msg == null) return; msg.setmsg(e.getMessage()); if (e.lnum>=0) { String s = ta.get
} } } if (end == -1) end = len; System.out.println("st/end: "+st+"/"+end); ta.select(st,end); //ta.replaceText("AAAAAH",st,end); // ta.selectAll(); ta.invalidate(); } } public boolean donep = true, errp =
runner.stop(); runner = null; } } public void stop() { if (runner !=null) { runner.stop(); runner = null; donep = true; if (netinitp == true) { client.out.flush(); netinitp = false; try {if (client.s != null) cli
workaround: store when key comes down, allow valid for // certain amount of time, if exceed then flush. autokey // events come in as keydown so should refresh naturally public void servicek
System.currentTimeMillis(); keys[ind] = true; } else if (n>='A'&&n<='Z') { int ind = n-'A'; keyt[ind] = System.currentTimeMillis(); keys[ind] = true; } return true; } public synchronized boolean keyU
stop(); // runp = true; progstr = s; start(); } public int executeprogram(String s) throws lerr { StringTokenizer st = new StringTokenizer(s,"\n"); Graphics g = getGraphics(); Rectangle bds = bounds
(Exception e) { e.printStackTrace(); return 1; } gpage(0); gpen(100); clearbits(); for(int i = 0; i<keys.length;i++) { keys[i] = false; keyt[i] = -1; } vect = new Vector(); vframes = new Vector(); PushFra
public void paint(Graphics g) { Rectangle r = bounds(); cb = bounds(); if (bw == 0) { createpage(100,100); bufim = createImage(r.width,r.height); } zoomr.reshape(0,0,24,16); drawframe(g); myg = g
PrintStream out; public ClientConnection(String host) { try { s = new Socket(host, PORT); in = new DataInputStream(s.getInputStream()); out = new PrintStream(s.getOutputStream()); System.ou
constraints = new GridBagConstraints(); String initcodestr = "{\n paper 50\n line 0 0 100 100\n}"; TextArea code; Button go = new Button("run"); Button frame = new Button("read files"); Butto
Button("step"); String infilename = null; Button saveb = new Button("save"); Button loadb = new Button("load"); public void setupforrun() { System.out.println("setup for run"); setLayout(new
BorderLayout()); p.setLayout(new BorderLayout());//new GridLayout(1,2)); Panel p3 = new Panel(); p3.setLayout(new BorderLayout()); code = new TextArea(initcodestr); p3.add("Center",code); ca
FlowLayout(FlowLayout.LEFT)); p2.add(new Label("Design by Numbers, Copyright 1998 John Maeda. All Rights Reserved.")); /*cbg = new CheckboxGroup(); { Checkbox c; cb1=c = new Checkbo
")); // p2.add(saveb); // p2.add(loadb); step.disable(); add("Center",p); // add("North",p2); // add("North",new accessorypanel(this,accessorypanel.tCMD)); p.add("Center",p3); p.add("West",canvas
true; } if (s!=null) { // run through and remove semicolons StringTokenizer f = new StringTokenizer(s,";"); initcodestr = ""; while(f.hasMoreTokens()) { initcodestr = initcodestr+f.nextToken()+"\n
name!=null) { try { initcodestr = readFile(infilename); } catch (Exception e) { initcodestr = ""; } if (runp) setupforrun(); else setupforedit(); } } public void autosimprep() { String s = code.getTex
spc; int cnt = 0; StringTokenizer st = new StringTokenizer(s,"\n"); while(st.hasMoreTokens()) { String t = st.nextToken(); t = t.trim(); spc = nsp*lev; if (t.length()>0) { if (t.charAt(0) == '{') { lev++;
s = code.getText(); canvas.runprogram(s); } public void doload() { Frame f = new Frame(); FileDialog fd = new FileDialog(f,"load",FileDialog.LOAD); fd.show(); String fl = fd.getFile(); if (fl!=null) {
msg.setmsg("done."); code.setText(new String(b,avail)); fp.close(); } catch (Exception e) { msg.setmsg("Can't load file "+fl); } } } public void dosave() { Frame f = new Frame(); FileDialog fd = n
msg.setmsg("Writing file: "+fl); ds.writeBytes(code.getText()); msg.setmsg("done."); fp.close(); } catch (Exception e) { msg.setmsg("Can't save file "+fl); } } } public boolean action(Event evt,
else if (evt.target == loadb) { return true; } else if (evt.target == step) { canvas.step(); return true; } else if (evt.target == auto) { return true; } else if (evt.target == stop) { canvas.stop(); return true
return true; } else return false; } public String picked() { String s = files.getSelectedItem(); String url = new String(); url = "http://acg.media.mit.edu/dbn/"+s+".html"; return url; } public String re
// InputStreamReader reader = new InputStreamReader(input); StreamTokenizer stream = new StreamTokenizer(input); int c; boolean first = false; stream.eolIsSignificant(true); stream.wordChar
(int)stream.nval; String str2 = Integer.toString(number); yippee = yippee + str2 + " "; break; case StreamTokenizer.TT_EOL: yippee = yippee + "\n"; break; case StreamTokenizer.TT_EOF: br
{ e.printStackTrace(); } return yippee; } public void paint(Graphics g) { } } import java.awt.*; import java.applet.*; import java.util.*; import java.net.*; import java.io.*; import java.lang.*; class ler
return b<len; } public StringTokenizerNoSkip(String ss) { s = ss;//.trim(); len = s.length(); } public String nextToken() { // looking for parens first, then brace, then space int i; boolean capturingp =
// each block contains vector of either Strings or a bonafide Block public class Block { Vector v; String s; int linenum; Vector args=null; // ugly hack to hide args for a define static int STR = 0, B
aa.charAt(i); if (c == '{') { lev++; res+=(c+""+lev); } else if (c == '}') { res+=(c+""+lev); lev—; } else res+=c; } if (lev !=0) throw new lerr("braces not balanced"); return res; } static int findstr(Vect
int i; Block b = new Block(); i = stind; b.setline(stind); // System.out.println("sten: "+stind+"/"+endind); while(i<endind) { String s = (String)v.elementAt(i); int len = s.length(); if (s!=null) { if (s.cha
else { Block bs = new Block(s); bs.setline(i); b.add(bs); i++; }} // } else i++; } return b; } static Block parse(String aa) throws lerr { Block b = new Block(); Vector pre = new Vector(); // System.out.
StringTokenizerNoSkip(aa); while(st.hasMoreTokens()) { String s = st.nextToken(); s = s.trim(); if (!s.equals("")) { pre.addElement(s); // System.out.println("add pre: "+s); } else pre.addElement(" "
s.startsWith(ss); } else return false; } public String getstr() { if (type==0) { return (" [S: "+s+"]"); } else { String a = " [B: "; int i; for(i=0;i<v.size();i++) { Block b = (Block)v.elementAt(i); a+=b.getstr
```

```
*; import java.util.*; import java.net.*; import java.io.*; import java.lang.*; import Block; import lerr; class accessorypanel extends Panel { // does double duty as top panel and bottom panel
    = false; public                ap, int type, bufpanel b) {          app = ap; bp = b; ty = type; ims = app.getImage(app.getDocumentBase(),dir+"stop.GIF"); imp =
=                  if (myg==null) myg =                  myg.setColor((t)>250?Color.gray:Color.darkGray); myg.drawLine(23,18,23+pw,18); } String msg="Hello."; public void
  } else {  bp.stop();          =          } if (myg==null)          =                  myg.drawImage(playp?ims:imp,4,4,this); } else if (loadr.inside(x,y)) { // System.out.println("loadr.....");
 null;          =                                                  donep: "+bp.donep); if (ty==tMSG) { loadr.reshape(r.width-48+4,0,24,r.height); saver.reshape(r.width-
                                                                g.setColor(Color.gray); g.fillRect(loadr.x+4,loadr.y+4,loadr.width-8,loadr.height-8);
 b = 0;      len;                                { s = ss.trim(); len = s.length(); } public String getuntil(int start, char c) { int i; for(i=start;i<len;i++) { if (s.charAt(i)==c) break; } if (c==' ') { b=i;
     level—;     (level == 0) break; } b=i+1; return s.substring(start,i+1); } public String nextToken() { // looking for parens first, then brace, then space int i; boolean capturingp = false;
          return getuntil(i,' '); } } return null; } } class bufpanel extends Panel implements Runnable { String hostname = "tschichold.media.mit.edu"; ClientConnection client; boolean
          int sc = 1; Hashtable cmdenv = null, ioenv = null; Graphics myg; boolean runtimep= false; TextArea ta = null; boolean bdirtyp = false; Color ltgr = new Color(239,239,239);
          bimg = bim.getGraphics(); p("buf size is: "+bw+" by "+bh); } boolean []keys = new boolean[26]; long []keyt = new long[26]; public void doinits() { int i; if(bitscol==null) { bitscol
          store lines in dynamically allocated linebuf public bufpanel(TextArea t,                m) { doinits(); ta = t; msg =                bdirtyp =          public
                g) { int i,j; Rectangle r = bounds(); cbw = bw*sc; cbh = sc*bh;                                                                                  if (runtimep) {
                for(i=0;i<16;i++)  for(j=0;j<16;j++) {  pg.setColor(((i+j)                                                                  } for(i=0;i<r.width/16+1;i++) for(j=0;j<r.height/16+1;j++)
                    // g.drawRect(framer.x-1,framer.y-1,framer.width+                                                                    1,cbh+1); g.setColor(Color.white); g.setFont(f); int
              x=framer.x+framer.width          y =                                                                                    g.setColor(Color.white);
    // drawbits(g); {  g.setColor(Color.black);                                                                                        g.drawString(sc+"x",zoomr.x+3,zoomr.y+12); } } public void
      = 1; } bdirtyp = true; } public void setmode(int m) { if (m== 0) sc = 1; else sc = 3; stop(); System.out.println("sc is          refreshbits(); repaint(); // refreshlines(); // drawslate(getGraphics()); }
        } public void gfield(int x0, int y0, int x1, int y1, int v) { int i,j; for(i=x0;i<=x1;i++) for(j=y0;j<=y1;j++)  setbit(i,j,100-v); } public void gcircle(int x0, int y0, int r) { int x = 0; int y = r; int d = 1-
x1, int y1) { gline(x0,y0,x1,y0); gline(x1,y0,x1,y1); gline(x1,y1,x0,y1); gline(x0,y1,x0,y0); } public void gline(int x0, int y0, int x1, int y1) { int dx = x1-x0; int dy = y1-y0; int i;
  (dx<0) { int dum = x0; x0 = x1; x1 = dum; } for(i=x0; i<=x1;i++) setbit(i,y0,pencol); } else { BresLine(x0,y0,x1,y1,pencol); /* if (dy>0) { if (dx>dy)  bresen1(x0,y0,x1,y1,pencol); else
  = x1-x0; int sny = (y1>y0)?1:-1; int dy = (y1-y0)*sny; int d = 2*dy-dx; int incrE = 2*dy; int incrNE = 2*(dy-dx); int x = x0; int y = y0; setbit(x,y,val); while(x<x1) { if (d<=0) {  d+=incrE;  x++; } else
    } public void setbit(int i, int j, int n) { if (i>bw-1) return; if (j>bh-1) return; if (j<0) return; if (i<0) return; // p("set: "+i+"/"+j+": "+n+" bds: "+bw+"/"+bh); bits[i][j] = n; dbits[i][j] = 1; j = bh-j;
                */          = true; } Color bitscol[] = null; boolean refreshp = false; Rectangle zoomr = new Rectangle(); public synchronized void drawbits(Graphics g) { int i,j; // int
                    g.setColor(Color.black);  g.drawImage(bim,0,0,bw*sc,bh*sc,this); g.translate(-framer.x,-framer.y); bdirtyp = false; } public void
              return ((Block)cmdenv.get(s)); } public Block get_io(String s) { return ((Block)ioenv.get(s)); } public void setvar(String s, int n) throws lerr { //
  if                {          st = new StringTokenizer2(trim2(s)); String intype = st.nextToken(); if (intype.equals("net")) {  if (netinitp == false) {  connecttoServer(hostname);  }
                Hashtable()); } public void PopFrame() { // System.out.println("-new frame popped"); vframes.removeElement(vframes.lastElement()); } Hashtable curFrame() {
            if ((m=(Integer)ht.get(s))!=null) break; } if (m == null) {  throw new lerr("Unknown variable: "+s); } else  return m.intValue(); } } public int contains(String s, char c) {
   }      void sp(String s) { // System.out.println(s); } // this is the io stack, it is a temporary kludge until // we build a universal stack. String instackp[] = new String[100]; int inc = 0; int out-
  "+v); outstack[outc] = v; outc++; } public String trim2(String s) { // remove start and end return s.substring(1,s.length()-1); } public void connecttoServer(String hoster) {
 place + " " + value); } public void sendtoServer(String nums) { client.out.println(nums); client.out.flush(); } public int getfromServer() { try { String line; line = client.in.readLine();  numgets++;
 // this is hard stuff // s = "A" // s = "A+1" // s = "(A+2)*B" // are there any operators? // if not, is it a number or symbol? // if yes, then stringtokenizer (evalarg(token1)+evalarg(token2)) // st "+-
 (s.startsWith("[")) { int v; String ss = trim2(s); System.out.println("ss is: "+ss); StringTokenizer2 st = new StringTokenizer2(trim2(s)); int x = evalarg(st.nextToken()), y = evalarg(st.nextToken());
      evalarg(s.substring(0,i))+evalarg(s.substring(i+1,s.length())));  case '*': return evalarg(s.substring(0,i))*evalarg(s.substring(i+1,s.length()));  case '-': return evalarg(s.substring(0,i))-
            st = new StringTokenizer(s, "+"); return evalarg(st.nextToken())+evalarg(comesafter(s, contains(s, '+')+1)); } else if (contains(s, '-') > 0) { StringTokenizer st = new
          contains(s, '*')+1)); } else if (contains(s, '/') > 0) { StringTokenizer st = new StringTokenizer(s, "/"); return evalarg(st.nextToken())/evalarg(comesafter(s, contains(s, '/')+1)); } else if
       { StringTokenizer2 st = new StringTokenizer2(trim2(s)); String intype = st.nextToken(); // System.out.println("THIS IS THE <>"); if (intype.equals("loc")||intype.equals("mouse")) { int
          if (num>=1 && num<=26) {  return keys[num-1]?100:0;  } else {  throw new lerr("Unknown <key>: "+num);  } } else if ( intype.equals("time")) { int num = evalarg(st.nextToken());
  if (intype.equals("net")) {  if (netinitp == false) {  connecttoServer(hostname);  } sendtoServer(st.nextToken());  return getfromServer(); } else {  Block gb;  Graphics g = this.getGraphics();  if
      for(int i=0;i<v.size();i++) {  setvar((String)v.elementAt(i),evalarg(st.nextToken()));  } doBlock(g,gb);  PopFrame();  return outputpop();  }  } } } else if (Character.isDigit(s.charAt(0))) { return
          Block b) throws lerr { String line = b.s; StringTokenizer2 st2 = new StringTokenizer2(line); String cmd = st2.nextToken(); int i; curt = System.currentTimeMillis(); //
                                                                                                                                  } else if (cmd.equals("cube")) { gpen(50); System.out.println("pen 50"); for(i=0;i<40;i++) {
   "+(60+i)+"          }                                  75");          for(i=0;i<40;i++) {  gline(60,20+i,80,40+i); System.out.println("line "+(60)+" "+(20+i)+" "+(80)+" "+(40+i)); } } else if
                { // for circle, should get absolute value      r, also rectangle // and field should be not order dependent.
      wt*=10; while (System.currentTimeMillis()-t<wt); } else if (cmd.equals("escape")) { return true; } else if (cmd.equals("field")) {
  if (cmd.equals("paper")) { gpage(evalarg(st2.nextToken())); } else if (cmd.equals("msg")) { System.out.println("**"+line); } else if (cmd.equals("pen")) { gpen(evalarg(st2.nextToken())); } else {
    Vector v = bb.args;  if (v == null) {  // no args  doBlock(g,bb);  } else {  PushFrame();  for(i=0;i<v.size();i++) {  setvar((String)v.elementAt(i),evalarg(st2.nextToken()));  }  doBlock(g,bb);
  <0) { for (int i=n; i>=t; i+=inc) { setvar(a, i); doBlock(g, b); } } else { for (int i=n; i<t+1; i+=inc) { setvar(a, i); doBlock(g, b); } } } // return 0 if normal // return 1 if get an 'escape' public boolean
== Block.BLK) { Vector v = b.v; for(i=0;i<v.size();i++) {  Block a = (Block)v.elementAt(i);  //System.out.println("DOING BLOCK: "+a.getstr());  if (a.isa("repeat")) {  //stepone A, 1, 5  //
  int start = evalarg(st.nextToken());  int end = evalarg(st.nextToken());  if (start>end) {  inc = -1;  }  doStep(g, arg, start, end,inc,nextb);  } else if (a.isa("forever")) {  Block nextb =
need robust image buffering? stack?  // yes, this happpened. have to disable this cool  // feature unfortunately  // but i need it because it looks great so try to fix it tom  // doBlock(g2,nextb);  if
          st = new StringTokenizer2(line); st.nextToken();  String arg1 = st.nextToken();  String arg2 = st.nextToken();  Block nextb = (Block)v.elementAt(i+1); i++;  if (evalarg(arg1) ==
   nextb = (Block)v.elementAt(i+1); i++;  if (evalarg(arg1) != evalarg(arg2))  doBlock(g,nextb);  } else if (a.isa("within?")) {  String line = a.s;  StringTokenizer2 st = new StringTokenizer2(line);
        } else if (a.isa("smaller?")) {  String line = a.s;  StringTokenizer2 st = new StringTokenizer2(line); st.nextToken();  String arg1 = st.nextToken();  String arg2 = st.nextToken();  Block
 arg1 = st.nextToken();  String arg2 = st.nextToken();  Block nextb = (Block)v.elementAt(i+1); i++;  if (evalarg(arg1) >= evalarg(arg2))  doBlock(g,nextb);  } else if (a.isa("command")) {  String line
 for(j=2;j<nargs;j++) {  String s3 = st.nextToken();  nv.addElement(s3);  }  nextb.args = nv;  }  def_command(arg,nextb);  //return false;  } else if (a.isa("number")) {  String line = a.s;  int nargs =
  {  String s3 = st.nextToken();  nv.addElement(s3);  }  nextb.args = nv;  }  def_io(arg,nextb);  } else {  doBlock(g, a);  } } } return false; } public String preprocess(String s) { return s; } public int
==null) { runner = new Thread(this); runner.start(); } } String progstr = null; int mx, my, mb; int orgx, orgy; public void p(String s) { System.out.println(s);} public void calcmouse() { mx =
calcmouse(); return true; } public boolean mouseExit(Event ev, int x, int y) { mousex = x; mousey = y;mouseb = false; calcmouse(); return true; } public boolean mouseDown(Event ev, int x, int
                start/stop"); if (runner==null) start(); else stop();//runprogram(progstr); } return true; } public boolean mouseUp(Event ev, int x, int y) { mousex = x; mousey = y;mouseb =
  mousey = y;mouseb = false; calcmouse(); return true; } public void step() { stepp = true; } public void msgbox(String s) { if (msg!=null) msg.setmsg(s); } public void msgalive() { if (msg!=null)
      int lnum = e.lnum; int st=-1, end=-1; int lc = 0; if (lnum ==0) st = 0; for(int i=0;i<len;i++) {  if (s.charAt(i) == '\n') {  lc++;  if (lc == lnum)  st = i+1;  else if (lc == lnum+1) {  end = i;  break;
run() { errp = donep = false; try { msgbox("Running..."); executeprogram(progstr); donep = true; msgbox("Done."); runner.stop(); runner = null; } catch (lerr e) { errp = true; msgbox(e);
 (IOException e2){}; } msgbox("Stopped."); //unp = false; } } // there is a bug in how key is trapped by win java (expect // worse in mac java). does not match keydowns with keyups //
          { if (keys[i]) {  if (curt-keyt[i]>1000) {  keyt[i] = -1;  keys[i] = false;  } } } } public synchronized boolean keyDown(Event ev, int n) { if (n>='a'&&n<='z') { int ind = n-'a'; keyt[ind] =
  (n>='a'&&n<='z') { int ind = n-'a'; keyt[ind] = -1; keys[ind] = false; } else if (n>='A'&&n<='Z') { int ind = n-'A'; keyt[ind] = -1; keys[ind] = false; } return true; } public void runprogram(String s) {
  through and remove comments // s = preprocess(s); // currently absorbed by block.parse incorrectly // 2. do syntax check into blocks try { bb = Block.parse(s); } catch (lerr e) { throw e; } catch
 Hashtable(); ioenv = new Hashtable(); // bb.pp(); // 4. do runblock try { doBlock(g,bb); } catch (lerr e) { throw e; } catch (Exception e) { e.printStackTrace(); return 1; } return -1; } Image bufim;
      } public Dimension preferredSize() { return new Dimension(szw,szh); } } class ClientConnection extends Object { public static final int PORT = 6800; Socket s; DataInputStream in;
   to          on "+PORT); } catch (IOException e) { System.out.println(e.toString()); } } } public class env extends Applet { // GridBagLayout layout = new GridBagLayout(); // GridBagConstraints
          Choice files = new Choice(); bufpanel canvas; int[] numb = new int[4]; CheckboxGroup cbg; Checkbox cb1,cb2; Button stop = new Button("stop"); Button step = new
    = new bufpanel(initcodestr,1); add("Center",canvas); // bufpanel.set } accessorypanel msg; public void setupforedit() { Panel p = new Panel(); Panel p2 = new Panel(); setLayout(new
          msg = new accessorypanel(this,accessorypanel.tMSG,canvas); canvas.msg = msg; p3.add("South",msg); code.setFont(new Font("Courier",Font.PLAIN,12)); p2.setLayout(new
          cb2=c = new Checkbox("zoom",cbg,false); p2.add(c); } */ // p2.add(go); // p2.add(step); // p2.add(stop); // p2.add(new Label("special  ")); // p2.add(auto); // p2.add(new Label(" i/o:
      s;          runp = false; Date d = new Date(); Date d2 = new Date(98,6,15); infilename = getParameter("url"); s = getParameter("initcode"); if (getParameter("run")!=null) { runp =
        initted:"); System.out.println(initcodestr); System.out.println("done."); if (runp) {  System.out.println("setting up for run");  setupforrun(); } else  setupforedit(); } else if (infile-
if (s.startsWith("@")) s = s.substring(3,s.length()-1); ns = s.replace(';','\n'); code.setText(ns); } public void autotabbify() { String s = code.getText(); String ns = ""; int nsp = 2; int i; int lev = 0; int
  == ')') {          spc-=nsp;  } } for(i=0;i<spc;i++)  t=" "+t; if (cnt == 0) ns = t; else ns = ns+"\n"+t; cnt++; } code.setText(ns); } public void goaheadandrun() { autosemiprep(); autotabbify(); String
   fp = new                DataInputStream ds = new DataInputStream(fp);  msg.setmsg("Reading file: "+fl);  int avail = ds.available();  byte b[] = new byte[avail];  ds.readFully(b);
  "                fd.show(); String fl = fd.getFile(); if (fl!=null) {  try {  FileOutputStream fp = new FileOutputStream(fl);  DataOutputStream ds = new DataOutputStream(fp);
          +code.getSelectionStart()); if (evt.target == go) { // here i do some insanely bad things // like autotab, each runcycle ... return true; } else if (evt.target == saveb) { return true; }
 ==       { // lines canvas.setmode(0); return true; } else if (evt.target == cb2) { // dots canvas.setmode(1); return true; } else if (evt.target == files) { readFile(picked()); //readFile(mainfr.picked());
          = new String(""); try { URL u = new URL(s); URLConnection conn = u.openConnection(); InputStream input = u.openStream(); //DataInputStream data = new DataInputStream(input);
          {  switch (c = stream.nextToken()) {  case StreamTokenizer.TT_WORD:  yippee = yippee + stream.sval + " ";  break;  case StreamTokenizer.TT_NUMBER:  int number;  number =
          //code.replaceText(yippee, 0, 10000); } catch (IOException e) {  e.printStackTrace(); } } catch (MalformedURLException e) { System.out.println("bad url"); } catch (IOException e)
 { int lnum =      public lerr(String s) { super(s); } public lerr(String s, int a) { super(s); lnum = a; } } class StringTokenizerNoSkip { String s; int b = 0; int len; public boolean hasMoreTokens() {
""; if (b == len) return null; i = b; while(i<len) { char c = s.charAt(i); // System.out.println(i+"/"+c); if (c == '\n') { i++; break;} else { s2+=c; i++; } } b = i; return s2; } } // program is a bunch of blocks
type; // 0 is string, 1 is block. void setline(int l) { linenum = l; } static String labelled(String aa) throws lerr { int i, len = aa.length(); int lev = 0; String res = ""; for(i=0;i<len;i++) { char c =
 int i; for(i=j;i<v.size();i++) { String s = (String)v.elementAt(i); if (s.charAt(1)==c)  return i; } return v.size(); } static Block doparse(Vector v, int stind, int endind) throws lerr { // start with 1st line
  c = s.charAt(1);  i++;  int targ = findstr(v,i,c);  b.add(doparse(v,i,targ));  i = targ+1; } else if (s.charAt(0) == '/') { // comment  if (s.charAt(1) == '/')  // do nothing  i++; } else if (s.length()<2) {  i++; }
at\n("+aa+")"); aa = aa.toLowerCase(); // label blocks try { aa = labelled(aa); } catch(lerr l) { throw l; } //System.out.println("after labelled\n"+aa); StringTokenizerNoSkip st = new
          } return doparse(pre,0,pre.size()); } public Block(String ss) { s=ss; type = STR; } public void sp(String s) { System.out.println(s); } boolean isa(String ss) { if (type == STR) { return
a; } } public void pp() { System.out.println(getstr()); } public Block() { v = new Vector(); type = BLK; } public void add(Block b) { v.addElement(b); } } // End. Let's hope this all works out somehow.
```

1 BEGIN Our forefathers at the Bauhaus, Ulm, and many other key centers for design education around the world labored to create a sense of order and method to their teaching. Thanks to their trailblazing work, teaching design at the university level gradually became accepted as a meaningful and constructive activity. A drawing board, small or large, became the stage for paper, pen, ink, and blade to interact in the disciplined activity that characterized the profession of visual design. Beginning in the early 1980s, however, affordable graphical computing systems emerged as the new drawing boards, forever disrupting the refined systems that had been carefully established over the previous decades. The page-description language PostScript was born, and the race for advanced software-based tools to aid a generation of computer-empowered youth began.

As could be expected by any sudden change in a field, the process of integrating computers into traditional design education has been akin to the proverbial mixing of oil and water. Understandably, the point-and-click ease of computers poses a threat to the painstakingly acquired skills of the pre-computer design educator. The fact that each new generation of students clearly recognizes its technical advantage over their instructors only widens the gap that separates past and present. Digitally adept faculty members are hired at an alarming rate to try to tame the young gurus, but even those instructors who profess a lead over students quickly lose their nominal superiority with each new software release. Unfortunately, most of the parties involved do not realize that computers, as they are used today, have nothing to do with design skill, or design education for that matter, but the computer industry strives to convince us otherwise.

2 COMMANDS Writing programs requires you to assume a forceful personality. The computer will do anything within its abilities, but it will do absolutely nothing unless commanded to do so. A command must be issued clearly, in the strict format of a one-word action, followed by a set of descriptors which qualify the action. There is no need to add the formality of a "please" or even a "thank you." A command should not be confused with a request; you can fully expect that it will be obeyed. Of course, if you were to address a co-worker with similar bluntness there would be ensuing personal problems. The computer, however, is quite comfortable with this direct manner of communication. The only time that the computer will complain is when it cannot perform the command you requested because either (1) the command is unknown to the computer, or (2) the way in which the command's descriptors are presented is not in an acceptable form.

Whenever the computer fails to cooperate, the novice usually accuses the computer of stupidity. But the computer cannot think beyond anything you tell it. In contrast, a fellow human can cope with situations of incomplete information and usually recover. For example, "Go ahead and do it." "Do what?" "You know." And usually that person knows. The computer does not yet have this level of intuitive comprehension; therefore, we must address it in a specific manner.

Tell the computer exactly what you want and the computer will comply without error. But you must state specifically and completely the action that you wish to execute.

Specify a Paper The first step is to see the software system we will use in this book. Go to the DBN Web site and connect to the main system. You should see a display similar to what appears here. On the right is the program editing area. On the left is the main display area where the visual output of the program will always be shown. In the middle of the display area there is a small square called the *paper,* where all of your drawing will take place. When you enter a program and press the run button, any subsequent processing will be shown within the paper area.

Before you can draw in the display area, you must first specify the shade of paper using the **Paper** command. There is only one kind of paper in the DBN system. This paper can be any shade of gray you desire, specified by a number of 0 to 100, where 100 is 100 percent black (solid black) and 0 is 0 percent black (white). Into the editing area, type "Paper 100" and run the following program:


```
Paper 100
```

A new 100 percent black sheet of paper should appear in the display. Depending upon the type of computer monitor you are using, the sheet is about 100 points square, which translates to about a 1.5-inch or 35-mm square. You may be concerned about these cramped dimensions, but you will find that mastering this small swatch of space is not trivial. Note that there is a space character between the Paper command and the numerical descriptor 100. You must always separate the command and descriptors with at least one space. A few examples of this simple program are displayed on the facing page.

Unlike real paper, the Paper command can be run as many times as you wish because there is never a shortage of virtual paper. The choices are either black, white, or the 99 shades of gray in between, which adds up to 101 possibilities.

Never type anything like "10 Paper" or "Paper10." In either of these cases, the computer tries to reconcile "10" or "Paper10" as a command and is unsuccessful because such words are not in the basic DBN vocabulary and will signal an error. The computer always requires that the first word on a program line be a command, followed by a set of descriptors. In the case of Paper, there is only one descriptor.

You must be careful to obey the specified format or the computer will not respond.

Paper 100

Paper 0

Paper 20

Paper 40

Paper 60

Paper 80

Paper 25

Paper 50

Paper 75

Paper 57

Paper 11

Paper 97

Paper 74

Paper 49

Paper 5

Paper 54

Paper 32

Paper 18

Paper 43

Paper 62

Paper 8

Paper 81

Paper 23

Paper 36

Paper 77

Paper 14

Paper 88

Selecting a Pen After choosing a paper to draw upon, you might guess that the second task to master is to select a pen. It might help to know it will be impossible to hold the pen.

When drawing, we tend to take for granted the natural reflex to extend our arm, grab a pen, and draw. This does not apply in the computational medium because the computer is both intermediary and medium. You instruct the computer to grab a pen, and then you instruct the computer what to draw upon itself.

The pen is selected with the **Pen** command and can draw in any shade of gray from 0 to 100, where 0 is white and 100 is solid black, for a total of 101 pens. Thus "Pen 100" selects a black pen.

```
Pen 100
```

When the program is run, there is no visible change to the paper because the Pen command only changes the internal state of the computer's readiness to draw. In contrast, the Paper command actually yielded a visible result, a new piece of paper in the requested shade.

You can specify a new sheet of paper and subsequently set the pen by issuing the two commands in sequence as the program:

```
Paper 100
Pen 0
```


As expected, there is no visible difference in the result, even though the pen is set. Notice that the commands are issued on separate lines of text. As a general rule, you should always enter commands on separate lines, not merely for legibility's sake, but because the syntax of the language requires you to do so.

Now that the paper is specified and pen is ready, all that is left is to draw something. But what? And how? For what purpose? The answers to the former two questions are revealed as this book progresses; the latter question is of utmost importance and can only be answered by you.

3 LINE Drawing a perfectly straight line usually requires a tool, such as a ruler. Although I have introduced analogues to physical implements such as paper and pen, there is no ruler. Instead there is the **Line** command for drawing a straight line that connects two points on the paper. Line will differ from Paper and Pen with respect to the quantity and type of descriptors needed. Paper and Pen require a single numerical value that specifies the percentage of black; Line requires a total of four numerical values that describe the locations of the two points to connect. Using numbers to specify tone or position should be familiar to professional visual creatives as part of their daily activity; however, even for those who are unfamiliar, the practice can be easily mastered.

By starting with a line instead of a freestyle stroke, you might guess that the drawings you create will be primarily of a geometric nature. Whenever you fail to see any beauty in these exercises and start to feel constrained, remember that the ideas presented are primarily of a structural nature and are mere building blocks to be formed into larger aspirations. All of the exercises have been designed to illustrate basic properties and skills related to computational media design which, given a great deal of determination, can be applied to realize any expressive intentions.

If you do not like what is being drawn, try to draw something else. Never let the computer suppress your will to freely express.

Describe a Line With Paper ready and Pen poised, the next step is to use the **Line** command. Specifying a line's starting and ending points is trivial if you point directly at the computer screen with your finger or indirectly using the mouse; however, these options are not available in this system. A line must be described as a corresponding line of text in the program editing area. Thus the format of that text needs to be clarified. Before revealing exactly how the Line command must be used, let us step back for a moment and think of how a line can be described.

Given the line within the frame to the left, how would you describe it verbally? There are a variety of possible answers ranging from the extremely vague to the proper and precise.

1 *Line from the bottom left to the upper right.*

2 *Line from a little from the left and less than a half from the bottom, to less than a half from the top and on the right edge.*

3 *Line from a third from the bottom and close to the left, to a third from the top on the right edge.*

4 *Line from a third from the bottom and close to the left, to two-thirds from the bottom exactly on the right edge.*

5 *Line from about 3/8 inch from the left and about 7/16 inch from the bottom, to about 7/16 inch from the top on the right edge.*

6 *Line from about 3 millimeters from the left and about 10 millimeters from the bottom, to about 11 millimeters from the top on the right edge.*

7 *Line from 3 millimeters from the left and 30 points from the bottom, to about 7/16 inch from the top on the right edge.*

8 *Line from 10 points from the left and 30 points from the bottom, to about 11 millimeters from the top on the right edge.*

9 *Line from 10 points from the left and 30 points from the bottom, to about 7/16 inch from the top on the right edge.*

10 *Line from 90 points from the right and 30 points from the bottom, to 30 points from the top and 0 points from the right edge.*

11 *Line from 70 points from the top and 10 points from the left, to 70 points from the bottom and 100 points from the left edge.*

12 *Line from 10 points from the left and 30 points from the bottom, to 30 points from the top and 100 points from the left edge.*

In general, the computer is terrible at comprehending such a wide variety of descriptive formats, but excellent at understanding a single consistent format. Humans can be similar in this respect, so there should be some sympathy to the computer's needs for a standardized method of description.

Numerical specifications are clear when they are exact and in like units. The roman typographic convention of the point system is convenient for the purposes of this discussion. Narrowing the set of possibilities to descriptions in points reduces the list to a smaller subset.

1 *Line from 90 points from the right and 30 points from the bottom, to 30 points from the top and 0 points from the right edge.*

2 *Line from 70 points from the top and 10 points from the left, to 70 points from the bottom and 100 points from the left edge.*

3 *Line from 10 points from the left and 30 points from the bottom, to 30 points from the top and 100 points from the left edge.*

When making measurements, choosing a reference of a closest edge versus the opposite edge is a natural habit of efficiency. For example, "10 points from the left" is quicker to physically measure than "90 points from the right." This convention requires that the reference be stated explicitly. Imposing a standard reference of the left and bottom edges for any horizontal and vertical position, respectively, removes the need to specify a distinction of reference, at the slight expense of initially being counterintuitive.

1 *Line from 10 points horizontal and 30 points vertical, to 70 points vertical and 100 points horizontal.*

2 *Line from 30 points vertical and 10 points horizontal, to 70 points vertical and 100 points horizontal.*

3 *Line from 10 points horizontal and 30 points vertical, to 70 points vertical and 100 points horizontal.*

The next convention to remove is the need to distinguish between horizontal and vertical dimensions by enforcing that the horizontal always be written before vertical, and that the two points be listed one after the other. The reference to the unit system of points becomes unnecessary and the result is narrowed to only one possibility.

1 *Line from 10 30 to 100 70.*

Finally, the prepositions and punctuation are removed, and the true form of the Line command is at last revealed.

```
Line 10 30 100 70
```

This format will seem cryptic at first, but with practice you can adapt to this concise description of a line. To build familiarity, get into the habit of reading the statement out loud as, for example, "Line from 10 over and 30 up, to 100 over and 70 up." Note that this particular line, and for that matter any other line, can be redescribed as an alternative where the order of the two pairs of number is reversed.

```
Line 100 70 10 30
```

A line is described by two pairs of numbers that represent two positions on the paper. The order of the pairs has no significance, but the order within a pair does. Each pair of numbers represents a set of dimensions that are always referenced from the left and bottom edges of the paper, where the horizontal measure always precedes the vertical.

Draw a Line You can now write a three-line program, which is your first major graphics program, if only a single black line on white paper. (Note: all drawings are rendered at half their original scale for enhanced page balance.)

```
Paper 0
Pen 100
Line 40 20 80 60
```

The line is 40 over and 20 up, to 80 over and 60 up—not a particularly exciting line, aside from the similar lack of smoothness in the opening example. The reasons for this jaggy characteristic will be discussed in Chapter 8, but for now understand that the paper we draw upon (not to be confused with the actual paper used in this book) is coarse in a virtual sense. Therefore, any mark left on the paper will have a rough, textural flavor, the classic signature of digital artwork. In some cases, the texture will not be apparent, such as in a perfectly horizontal or vertical line. In these cases, the lines are smooth because they essentially fit into the perfect horizontal and vertical grooves in the texture of the page. But in general, the lines will be coarse.

You can now draw a variety of lines on different papers. The concept is best illustrated through practice. A good introductory exercise is to inspect the adjacent compositions by looking at the picture and imagining the corresponding program, or even better, looking at the program and imagining the corresponding picture.


```
Paper 100
Pen 1
Line 0 83 56 27
```



```
Paper 20
Pen 100
Line 100 0 10 11
```



```
Paper 50
Pen 0
Line 0 5 100 4
```



```
Paper 0
Pen 20
Line 40 40 90 90
```



```
Paper 15
Pen 100
Line 48 23 75 80
```



```
Paper 80
Pen 60
Line 58 0 50 84
```



```
Paper 0
Pen 100
Line 24 99 24 8
```

Where Is the Line? There are some cases where you draw a line and do not see the intended result. For instance, imagine what happens when you draw outside the paper?

```
Paper 0
Pen 100
Line 20 30 200 300
```

The line starts from 20 left and 30 up to 200 left and 300 up, but you never see the line as it leaves the 101-point square. Any portion of a line drawn outside the space will be ignored and clipped to the edges in this fashion.

Also, what happens when you draw in the same shade as the paper?

```
Paper 66
Pen 66
Line 0 0 100 100
```

Since the digital medium is exact, drawing a diagonal line in the same shade as the underlying paper appears to produce nothing, even though you might expect to see some faint impression of the line as is common in a double hit of ink.

Another surprise occurs when you draw a line and then request a new sheet of paper.

```
Line 0 50 100 50
Paper 50
Pen 0
```

In this case, the intent is to draw a white horizontal line across the center of a gray sheet of paper, but the new sheet of paper is placed on top of the line. Thus the line is not visible.

Finally, there is the case where the Pen is not explicitly set and a Line is drawn.

```
Paper 100
Line 3 33 97 66
```

When the Pen is not set, the default setting is 100 percent black. As an additional convention, when the Paper is not set explicitly, the default setting is 0 percent black, meaning white.

With the exception of the first example, nothing appears to happen in the visual output. But be aware that something has indeed happened from the viewpoint of the computer. The computer has labored to create a line, just as if the line were visible. The computer does not know when it has done something completely useless, such as drawing a black line on black paper. Only you do.

Summary The process of drawing requires a means to establish where the pen is to be placed on the paper. Otherwise, the computer cannot know where to draw. Your hands usually serve as the means to locate points in space; however, in the computational model of drawing, the only purpose your hands serve is to type commands. The constraint of measuring horizontal and vertical dimensions from the lower-left corner was introduced as a way to make dimensions consistent. Dimensions are always in units of points.

The Line command is followed by two points, where each point is a pair of numbers, always horizontal then vertical, and separated by spaces.

Draw More Than One Line Drawing two lines is as simple as drawing a single line, only it's twice the work.

```
Paper 0
Pen 100
Line 50 0 50 100
Line 0 50 100 50
```

You can change the shade of the second Line by changing the pen value with another invocation of the Pen command.

```
Paper 0
Pen 30
Line 0 50 100 50
Pen 100
Line 50 0 50 100
```

Lines are drawn in the order that the Line commands are issued, which you can verify when lines intersect. The difference is realized by simply rearranging the order in which Pen and Line commands are used.

```
Paper 0
Pen 100
Line 50 0 50 100
Pen 30
Line 0 50 100 50
```

The gray line overlaps the black line at the center, whereas a change in the drawing order will cause the reverse to occur, shown here magnified:

3x mag

Due to the effect of overlap, you can easily create the illusion of three lines from just two lines.

```
Paper 100
Pen 0
Line 30 30 100 30
Pen 50
Line 60 30 80 30
```

But it is also easy to mistakenly create one line from two lines.


```
Paper 0
Pen 50
Line 40 50 60 50
Pen 100
Line 30 50 70 50
```

Draw More Than Two Lines Three lines render the familiar shape of a triangle.


```
Paper 80
Pen 20
Line 33 33 33 66
Line 33 66 66 66
Line 66 66 33 33
```

```
Paper 0
Pen 80
Line 35 30 65 30
Line 65 30 65 70
Line 65 70 35 30
```



```
Paper 100
Pen 0
Line 50 77 22 27
Line 22 27 78 27
Line 78 27 50 77
```



```
Paper 33
Pen 66
Line 77 0 88 7
Line 88 7 10 96
Line 10 96 77 0
```

Four lines compose a square, five a star, six a hexagon, and seven a floating pyramid.


```
Paper 20
Pen 50
Line 30 30 70 30
Line 70 30 70 70
Line 70 70 30 70
Line 30 70 30 30
```



```
Paper 33
Pen 78
Line 20 54 80 54
Line 80 54 32 20
Line 32 20 50 77
Line 50 77 68 20
Line 68 20 20 54
```

```
Paper 20
Pen 70
Line 32 61 32 39
Line 68 61 68 39
Line 68 61 50 71
Line 32 61 50 71
Line 32 39 50 29
Line 68 39 50 29
```

```
Paper 0
Pen 100
Line 40 50 70 35
Line 20 30 50 15
Line 70 35 50 15
Line 40 50 20 30
Line 40 50 50 70
Line 70 35 50 70
Line 20 30 50 70
```

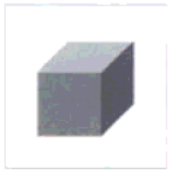
An increased number of lines results in a transition from abstract to the representational.

```
Paper 0
Pen 100
Line 53 72 50 78
Line 50 78 54 82
Line 54 82 57 81
Line 57 81 59 77
Line 59 77 56 73
Line 56 73 53 72
Line 49 70 50 46
Line 58 69 60 45
Line 49 70 60 69
Line 49 70 42 63
Line 42 63 33 74
Line 60 69 64 57
Line 64 57 64 46
Line 50 46 60 45
Line 50 45 46 31
Line 46 31 49 13
Line 60 45 61 29
Line 61 29 63 11
```

```
Paper 0
Pen 100
Line 0 30 23 30
Line 23 30 23 40
Line 23 40 17 39
Line 17 39 12 46
Line 12 46 16 52
Line 16 52 15 57
Line 15 57 27 58
Line 27 58 34 52
Line 34 52 33 44
Line 33 44 26 40
Line 26 40 27 30
Line 27 30 40 30
Line 40 30 38 44
Line 36 43 55 58
Line 55 58 76 43
Line 73 45 72 30
Line 72 30 100 30
Line 46 30 47 30
Line 47 30 46 40
Line 46 40 53 40
Line 53 40 53 30
Line 53 30 55 30
Line 59 40 67 40
Line 67 40 66 35
Line 66 35 59 35
Line 59 35 59 40
```



```
Paper 20
Pen 100
Line 100 8 25 22
Line 20 25 50 36
Line 56 38 68 42
Line 20 25 20 18
Line 20 18 90 5
Line 44 14 44 0
Line 50 8 50 0
Line 54 11 54 0
Line 51 34 56 35
Line 50 35 35 60
Line 50 40 41 55
Line 54 40 41 60
Line 34 61 64 80
Line 41 60 61 75
Line 68 86 73 80
Line 68 86 65 84
Line 73 80 70 78
Line 63 83 67 75
Line 67 75 66 65
Line 66 65 82 75
```



```
Paper 0
Pen 50
Line 20 20 60 20
Line 20 21 60 21
Line 20 22 60 22
Line 20 23 60 23
Line 20 24 60 24
Line 20 25 60 25
Line 20 26 60 26
Line 20 27 60 27
Line 20 28 60 28
Line 20 29 60 29
Line 20 30 60 30
Line 20 31 60 31
Line 20 32 60 32
Line 20 33 60 33
Line 20 34 60 34
Line 20 35 60 35
Line 20 36 60 36
Line 20 37 60 37
Line 20 38 60 38
Line 20 39 60 39
Line 20 40 60 40
Line 20 41 60 41
Line 20 42 60 42
Line 20 43 60 43
Line 20 44 60 44
Line 20 45 60 45
Line 20 46 60 46
Line 20 47 60 47
Line 20 48 60 48
Line 20 49 60 49
Line 20 50 60 50
Line 20 51 60 51
Line 20 52 60 52
Line 20 53 60 53
Line 20 54 60 54
Line 20 55 60 55
Line 20 56 60 56
Line 20 57 60 57
Line 20 58 60 58
Line 20 59 60 59
Pen 25
Line 20 60 60 60
Line 21 61 61 61
Line 22 62 62 62
Line 23 63 63 63
Line 24 64 64 64
Line 25 65 65 65
Line 26 66 66 66
Line 27 67 67 67
Line 28 68 68 68
Line 29 69 69 69
Line 30 70 70 70
Line 31 71 71 71
Line 32 72 72 72
Line 33 73 73 73
Line 34 74 74 74
Line 35 75 75 75
Line 36 76 76 76
Line 37 77 77 77
Line 38 78 78 78
Line 39 79 79 79
Pen 75
Line 60 20 80 40
Line 60 21 80 41
Line 60 22 80 42
Line 60 23 80 43
Line 60 24 80 44
Line 60 25 80 45
Line 60 26 80 46
Line 60 27 80 47
Line 60 28 80 48
Line 60 29 80 49
Line 60 30 80 50
Line 60 31 80 51
Line 60 32 80 52
Line 60 33 80 53
Line 60 34 80 54
Line 60 35 80 55
Line 60 36 80 56
Line 60 37 80 57
Line 60 38 80 58
Line 60 39 80 59
Line 60 40 80 60
Line 60 41 80 61
Line 60 42 80 62
Line 60 43 80 63
Line 60 44 80 64
Line 60 45 80 65
Line 60 46 80 66
Line 60 47 80 67
Line 60 48 80 68
Line 60 49 80 69
Line 60 50 80 70
Line 60 51 80 71
Line 60 52 80 72
Line 60 53 80 73
Line 60 54 80 74
Line 60 55 80 75
Line 60 56 80 76
Line 60 57 80 77
Line 60 58 80 78
Line 60 59 80 79
```

Because of the effort required to draw on the computer in this laborious way, where each line corresponds to four numbers that must be input, returning to conventional pen and paper would appear to be the more attractive option. Indeed, manually inputting a stream of numbers is insane when the same job could be done much better with regular pen and paper.
But there are superior methods of drawing on the computer that will demonstrate less emphasis on manual input and more emphasis on a style of *automatic* input.

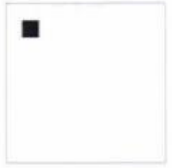
Comments Before more advanced programs can be examined, the issue of program legibility must be addressed. When a program sequence is long, ensuring that the commands are executed in an order that matches your intentions is not a trivial task. While writing a program, you can usually remember which program line corresponds to which visual result; however, if you don't use the program for a few days, you might get lost trying to decipher your own code. It can also be difficult for another person to follow your programming intentions when facing many lines of opaquely defined code. Do the first 100 lines of code generate a tree or a house? Does the pen command mark the beginning of a new object? Although comments do nothing to affect the program, you can significantly increase the legibility of a program by interspersing textual comments.

Comments are added using the // command followed by an informative message. The message can be the title of your program, part of a step-by-step narrative of events in the program, or simply messages to yourself for the purpose of maintaining sanity during development of a long program. A comment is only valid on a line of its own, and cannot be combined with other programming statements.


```
// My First Clear Program
// Version 1
// This program makes a new,
// gorgeous sheet of black paper
Paper 100
```



```
// Set the paper type
Paper 20
// Set the pen
Pen 80
// Draw a line from the lower left
// corner to the upper right corner
Line 0 0 100 100
```



```
// What follows is ten lines ...
Paper 0
Pen 100
Line 10 80 20 80
Line 10 81 20 81
Line 10 82 20 82
Line 10 83 20 83
Line 10 84 20 84
Line 10 85 20 85
Line 10 86 20 86
Line 10 87 20 87
Line 10 88 20 88
Line 10 89 20 89
Line 10 90 20 90
// I wish there were an easier way to
// draw a square, or other filled
// rectangular areas ...
```

```
//
// Pause a bit to contemplate how
// nice an empty room looks, and
// consider how putting things into
// it might mess things up. But
// remember that you don't want
// you parents to worry, so you
// have to put something in there.

// ********* buy some furniture
// ********* paint it in white
Pen 0
// ********* a place to sit
Line 45 52 63 50
Line 45 52 38 42
Line 56 40 63 50
Line 38 42 56 40
// ********* needs some legs
Line 42 41 42 32
Line 53 39 53 31
Line 46 40 46 37
Line 58 42 58 36
// ********* something to lean against
Line 46 61 62 59
Line 46 61 46 69
Line 62 67 46 69
Line 62 67 62 59
// ********* have to support it
Line 54 51 54 59

//
// A tiny door in the wall (well,
// not really a door but a 'hole')
// is just big enough for a
// friend to come and go as he/she
// pleases.

// ********* a hole in the wall
Pen 20
Line 10 32 10 36
Line 10 36 12 40
Line 12 40 14 40
Line 14 40 14 38

//
// Comments aren't usually seen
// by other people, and thus you
// can ramble like this all you
// like. The fact that this will
// be published verbatim perhaps
// defeats the private nature of
// a good session of comments,
// but for the sake of education
// we must do all that we can.
```

Summary Because creating everyday drawings can involve hundreds of line commands, it may appear that writing programs to draw pictures is more suitable for simple subject matter. For five to ten lines, the advantage of numeric input is that the results are precise and are easy to edit to a high degree of satisfaction. Anything over ten lines can be difficult to manage. You would be better off drawing on regular paper with a regular pen instead of wrestling with a computerized Paper, Pen, and Line.

Then why write programs? First of all, programs make it possible to pinpoint position and tone to an exact specification. Second and most important, more advanced programs allow you to explore spaces of ten, one hundred, or one million lines with ease, which may seem unfathomable at this stage. Working toward an advanced goal usually implies greater complexity; however, in the case of programming, advancement is paradoxically in the direction of simplification.

Paper, Pen, and Line are the three commands that have been introduced thus far. You can use Line to effortlessly draw one, two, or many lines with unerring precision. Each line can be independently shaded to a specific percentage of black using the Pen command. You can also specify the background color with the Paper command. When there are many lines of code, interspersing comments for the reader or yourself is a useful technique in developing a clear program.

Comments are a means for the designer to leave a literal trail of thought as part of the process of creation on the computer.

Design By Numbers
John Maeda
foreword by Paola Antonelli

Most art and technology projects pair artists with engineers or scientists: the artist has the conception, and the technical person provides the know-how. John Maeda is an artist *and* a computer scientist, and he views the computer not as a substitute for brush and paint but as an artistic medium in its own right. *Design By Numbers* is a reader-friendly tutorial on both the philosophy and nuts-and-bolts techniques of programming for designers and artists.

Practicing what he preaches, Maeda composed *Design By Numbers* using a computational process he developed specifically for the book. He introduces a programming language and development environment, available on the Web, which can be freely downloaded or run directly with any JAVA-enabled Web browser. Appropriately, the new language is called DBN (for "design by numbers"). Designed for "visual" people—artists, designers, anyone who likes to pick up a pencil and doodle—DBN has very few commands and consists of elements resembling those of many other languages, such as LISP, LOGO, C/JAVA, and BASIC.

John Maeda is the Associate Director, Sony Career Development Professor of Media Arts and Sciences/Associate Professor of Design and Computation, and Director of the Aesthetics and Computation Group at the MIT Media Lab. With his wife, Kris Maeda, he runs a design studio in Lexington, Massachusetts.

"This may well be the first software manual that you'd actually volunteer to have lying around on your coffee table."

—Liz Bailey, *Daily Telegraph*

The MIT Press
Massachusetts Institute of Technology
Cambridge, Massachusetts 02142
http://mitpress.mit.edu

0-262-63244-6