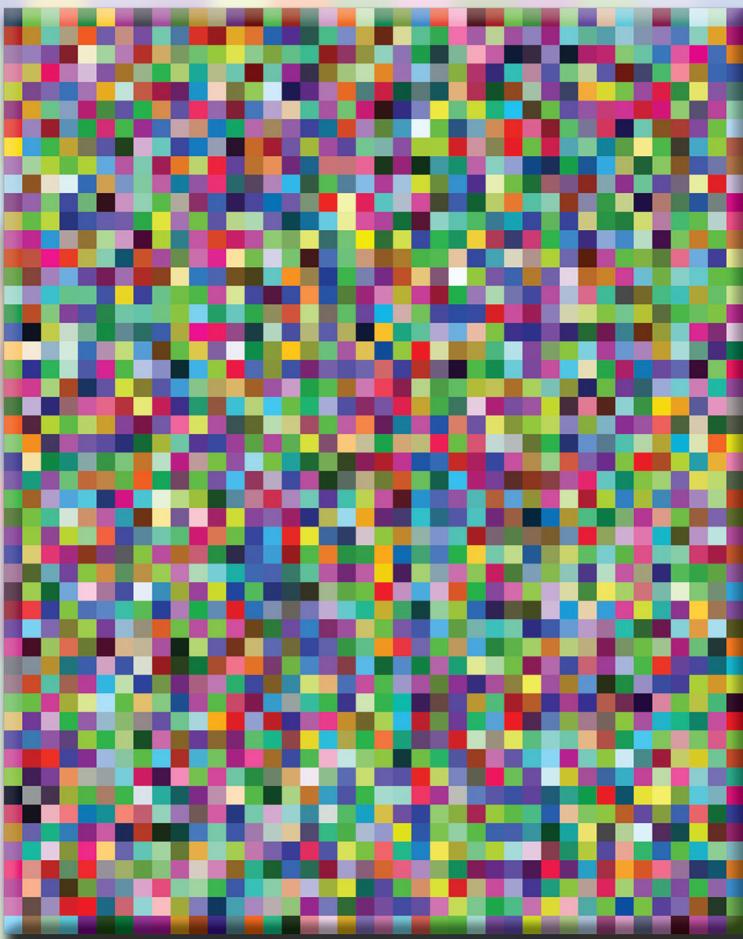


PROCESSING

An Introduction to Programming



**Jeffrey L. Nyhoff
Larry R. Nyhoff**



CRC Press
Taylor & Francis Group

A CHAPMAN & HALL BOOK

Pemrosesan: Sebuah Pengantar ke Pemrograman



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Pemrosesan: Sebuah Pengantar ke Pemrograman

oleh

Jeffrey L. Nyhoff

Larry R. Nyhoff



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Pers CRC
Grup Taylor & Francis
6000 Rusak Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2017 oleh Taylor & Francis Group, LLC
CRC Press adalah anak perusahaan Taylor & Francis Group, sebuah bisnis Informa

Tidak ada klaim atas karya asli Pemerintah AS

Dicetak pada kertas bebas asam

Buku Standar Internasional Nomor-13: 978-1-4822-5595-9 (Paperback)

Buku ini berisi informasi yang diperoleh dari sumber-sumber otentik dan terhormat. Upaya wajar telah dilakukan untuk mempublikasikan data dan informasi yang dapat diandalkan, namun penulis dan penerbit tidak dapat bertanggung jawab atas validitas semua materi atau konsekuensi penggunaannya. Penulis dan penerbit telah berupaya melacak pemegang hak cipta dari semua materi yang direproduksi dalam publikasi ini dan meminta maaf kepada pemegang hak cipta jika izin untuk menerbitkan dalam bentuk ini belum diperoleh. Jika ada materi hak cipta yang belum diakui, silakan tulis dan beri tahu kami agar kami dapat memperbaikinya dalam pencetakan ulang di masa mendatang.

Kecuali diizinkan berdasarkan Undang-Undang Hak Cipta AS, tidak ada bagian dari buku ini yang boleh dicetak ulang, direproduksi, dikirim, atau digunakan dalam bentuk apa pun dengan cara elektronik, mekanis, atau cara lain apa pun, yang sekarang diketahui atau yang akan ditemukan, termasuk fotokopi, mikrofilm, dan rekaman, atau dalam sistem penyimpanan atau pengambilan informasi apa pun, tanpa izin tertulis dari penerbit.

Untuk izin memfotokopi atau menggunakan materi secara elektronik dari karya ini, silakan akses www.copyright.com (<http://www.copyright.com/>) atau hubungi Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC adalah organisasi nirlaba yang menyediakan lisensi dan registrasi untuk berbagai pengguna. Bagi organisasi yang telah diberikan izin fotokopi oleh CCC, sistem pembayaran terpisah telah diatur.

Pemberitahuan Merek Dagang: Nama produk atau perusahaan mungkin merupakan merek dagang atau merek dagang terdaftar, dan hanya digunakan untuk identifikasi dan penjelasan tanpa maksud untuk melanggar.

Perpustakaan Kongres Mengkatalogkan Data dalam Publikasi

Nama: Nyhoff, Jeffrey, penulis. | Nyhoff, Larry R., penulis.
Judul: Pemrosesan : pengenalan pemrograman / Jeffrey L. Nyhoff, Larry R. Nyhoff.
Deskripsi: Boca Raton : CRC Press, 2017.
Pengidentifikasi: LCCN 2016041238 | ISBN 9781482255959 (pbk. : alk. paper)
Subjek: LCSH: Pengolahan (Bahasa program komputer) | Pemrograman komputer - Belajar dan mengajar.
Klasifikasi: LCC QA76.73.P75 N94 2017 | Catatan LC DDC 005.1071-dc23 tersedia di <https://lccn.loc.gov/2016041238>

Kunjungi situs Web Taylor & Francis di
<http://www.taylorandfrancis.com>

dan situs Web CRC Press di
<http://www.crcpress.com>

Kepada Sharlene, Rebecca, Megan, dan Sarakate.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Isi

Kata Pengantar, xv

Kata Pengantar: Mengapa Kami Menulis Buku Ini dan Untuk Siapa Buku Ini Ditulis, xvii

Ucapan Terima Kasih, xxi

Pendahuluan: Selamat Datang di Pemrograman Komputer, xxiii

Cbab1•Gambar Dasar dalam Pengolahan	1
Memulai Program Baru	1
Menyimpan Program	2
Mengambil Program	3
Memasukkan Kode ke dalam Gambar Dasar	4
Editor Teks dengan Elemen Grafis	8
Mengatur Ukuran “Kanvas”: Melihat Lebih Dekatukuran()Poin	8
Gambar Fungsi: Thetitik()Fungsi Menggambar Segmen Garis: The	10
garis()Fungsi Menggambar Persegi Panjang : Thelurus()Fungsi	11
Menggambar Elips: Theelips()Fungsi Menggambar Segitiga : The	13
segi tiga()Fungsi Menggambar Segi Empat : Thesegi empat()Fungsi	17
Menggambar Busur: Thebusur()Ringkasan Fungsi	20
	22
	26
	29
Referensi Pemrosesan Lebih	31
lanjut tentang Elemen Grafis	31
Urutan Penumpukan	32
Mengubah Ketebalan Garis: TheBerat pukulan()Fungsi	33
Bekerja dengan Warna: RGB	35
Menyetel Ulang Kanvas: Thelatar belakang()Fungsi	37

Mengubah Warna Isi: Themengisi()Dantanpa Isi()Fungsi Mengubah Warna Goresan : Thestroke()Dantidak ada Pukulan()Fungsi Komentar Sebaris	38
Skala abu-abu	43
Transparansi	45
Ringkasan	46
Latihan	47
Cbab2■Jenis, Ekspresi, dan Variabel	53
Nilai-nilai	53
Nilai Numerik	53
Bilangan bulat: Ituke dalamJenis	54
Bilangan dengan Poin Desimal: ThemengambangKetik	54
Aritmatika denganke dalamNilai danmengambangNilai-nilai ke dalamHitung	54
<i>Pembagian Bilangan Bulat</i>	55
<i>Menghitung Sisa dengan Operator Modulo.%</i>	56
mengambangHitung	57
<i>mengambangPecahan</i>	58
<i>Masalah Pecahan di Komputer</i>	59
Mengevaluasi Ekspresi	60
Urutan Operasi	63
Menggunakan Tanda Kurung	63
Variabel	65
Variabel yang Telah Ditentukan Sebelumnya:lebarDan tinggi Manfaat Menggunakan Variabel Membuat dan Menggunakan Variabel Sendiri Nama Variabel	67
	72
	73
	74
Jenis Variabel	75
Mendeklarasikan Variabel	76
Menetapkan Nilai ke Variabel Menggabungkan	77
Deklarasi dan Inisialisasi Menggunakan Kembali Variabel	80
Jenis Ketidakcocokan	80
Mengapa Tidak Gunakan Hanya mengambang	83
Jenis? Ekspresi dalam Pernyataan Penugasan	87
	88

Menggunakan Variabel di Sisi Kanan Pernyataan Penugasan Hati-hati dengan Pembagian Integer	91
Menugaskan Kembali Nilai Baru ke Konstanta Variabel	95
Konstanta yang Telah Ditentukan Sebelumnya	98
Mendefinisikan Jenis Konstanta	106
Nonnumerik Kita Sendiri	108
Karakter Individu: ThearangKetik	110
Beberapa Karakter: TheRangkaianJenis	114
RangkaianRangkaian	116
Ringkasan	118
Latihan	119
<hr/>	
Cbab3•Mbijih tentangkamumenyanyiPpmrosesanButilt-InFpengurapan	125
Lebih lanjut tentang Output Konsol: Themencetak()DancetakIn()Fungsi	125
Menampilkan Banyak Item ke Konsol	126
Teks Grafis dalam Pemrosesan	127
Ituteks()Fungsi	128
Ituukuran teks()Fungsi Itu	128
perataan teks()Fungsi	129
Mencocokkan Tipe Argumen dengan Tipe Parameter Dua Macam Fungsi	130
ruang kosongFungsi	132
Fungsi yang Mengembalikan Nilai	134
<i>Menentukan Tipe Pengembalian Fungsi Menggunakan Referensi</i>	137
<i>Pmrosesan Contoh: Menghitung Sisi Miring denganpersegi()Fungsi Itu kekuatan()Fungsi</i>	138
<i>Menghitung Luas Persegi Menggunakankekuatan()</i>	140
<i>Fungsi Menghitung Sisi Miring dengankekuan()</i>	141
<i>Fungsi Ituacak()Fungsi</i>	144
<i>Itubulat()Fungsi</i>	145
<i>Menggunakanbulat()Fungsi</i>	146
<i>Lebih Banyak Fungsi Konversi:ke dalam()Danmengambang()</i>	147
<i>ke dalam()Danacak()Contoh: Memilih Bilangan Bulat Acak untuk Mensimulasikan Pelemparan Sebuah Dadu</i>	148
<i>ke dalam()Danacak()Contoh: Panjang Sisi Acak untuk Persegi</i>	149

<i>Contoh: Memilih Lokasi Piksel Secara Acak untuk Lingkaran</i>	152
<i>Warna Acak</i>	156
<i>Itumengambang()Fungsi</i>	158
<i>Menggunakan mengambang()Fungsi: Membagi Duake dalam Variabel yang Secara Akurat Dikonversi ke Tipe Nonnumerik: The arang()Dan str()Fungsi Itu arang()Fungsi</i>	159
<i>Itu str()Fungsi</i>	162
Mensimulasikan Termometer Analog	163
Fungsi Khusus untuk Input dan Output dengan Kotak Dialog	164
<i>Input Menggunakan Kotak Dialog: The tampilkanDialogInput()Fungsi Memasukkan sebuah ke dalam atau mengambang()Nilai</i>	167
<i>Output Menggunakan Kotak Dialog: The tampilkanMessageDialog()Contoh Interaktif Fungsi: Menggambar Sudut</i>	168
Ringkasan	172
Latihan	174
<hr/> Cbab4■Pemrograman Bersyarat dengan jika	177
Analogi Resep	178
Dasar jika Penyataan	187
Dasar jika Contoh Pernyataan: Permainan Menggulirkan Die	188
Perhatian: Tidak Ada Titik Koma di Akhir Baris Pertama an jika Pernyataan Pemeriksaan	189
Kesetaraan: ke dalam Nilai-nilai	191
Dasar jika Pernyataan: Contoh Grafis The jika-lainnya Penyataan	192
Menggunakan jika-lainnya Pernyataan: Mengklasifikasikan	193
Usia Menggunakan jika-lainnya Pernyataan: Contoh Grafis The jika-lain-jika-lainnya Contoh Pernyataan Batu-Kertas-Gunting:	196
jika-lain-jika-lainnya	199
Menggunakan jika-lain-jika-lainnya Pernyataan: Contoh Grafis	200
Menggunakan jika-lain-jika-lainnya Pernyataan: Memperbarui Contoh Termometer	202
Menggunakan jika-lain-jika-lainnya Pernyataan: Jenis-Jenis Sudut Operasi Logika: AND (&&), OR ()	203
Penggunaan Logika AND (&&): Melempar Dua Dadu	209
Menggunakan Logika OR (): Melempar Sebuah Mati	220
Logis OR () Contoh: Melempar Dua Dadu	222
	223

BersarangjikaPernyataan	225
Menggunakan BersarangjikaPernyataan: Mengklasifikasikan	226
Usia booleanVariabel	229
Menggunakan sebuahbooleanVariabel untuk Menyimpan Kondisi	230
tersebutmengalihkanPernyataan	235
MenggunakanmengalihkanPernyataan: Meninjau Kembali Contoh Penggabungan Batu-Kertas-GuntingmengalihkanKasus: Menggulirkan Satu atau Enam Lainnya mengalihkan	235
Contoh Pernyataan: Hari dalam Bulan Memeriksa Kesetaraan:mengambangNilai-Nilai	239
Pengecekan Kesetaraan:RangkaianObjek tersebutsama dengan()Metode aRangkaian	240
Ringkasan Objek	243
	243
	246
	247
Latihan	248
<hr/> Cbab5•Pengulangan dengan Lingkaran:TDiaketikaPernyataan	255
Pengulangan Selama Kondisi Tertentu Benar	255
Pengulangan denganketikaPernyataan	256
MengunakanketikaPernyataan: Melemparkan Sebuah Dadu Berulang Kali	256
Menghindari Perulangan Tak Terbatas	262
Hati-hati dengan Titik Koma Ekstra!	263
MengunakanketikaPernyataan: Contoh Grafis	264
MengunakanketikaPernyataan: Simulasi Kartu Hadiah	273
MengunakanketikaPernyataan: Lingkaran Penghitungan untuk Menggambar Lingkaran Konsentris	279
Mengomentari/Membatalkan Komentar	284
MengunakanketikaPernyataan: Permainan Menebak Interaktif	284
Operator Logika BUKAN (!).	291
Menggunakan Logika NOT (!) Dengan aketikaKondisi Berhenti Loop	291
Berulang Denganlakukan sementaraRingkasan Pernyataan	293
	296
Latihan	296
<hr/> Cbab6•Membuat Loop Penghitungan MenggunakanuntukPernyataan	301
Penggunaan Loop Penghitungan	301
Persyaratan Loop Penghitungan	301
Membuat Loop Penghitungan dengan aketikaPernyataan	302
Membuat Loop Penghitungan dengan auntukPernyataan	305

Menelusuri AksiuntukPerhatian	307
Pernyataan: Hanya Dua Titik Koma!	310
Menghindari Perulangan Tak Terbatas	310
Menambah dan MengurangiuntukVariabel Penghitung Loop	311
Menggunakan Sama dengan dalam Kondisi Loop	313
Mengulangi Tindakan dengan Lingkup Lingkaran	314
PenghitunganuntukVariabel Penghitungan Pernyataan	316
Menghitung melalui Barisan dengan auntukPernyataan: Contoh Grayscale	318
Menghitung Barisan dengan auntukPernyataan: Meninjau Kembali Sebuah Contoh	321
Melakukan Tindakan Beberapa Kali Tertentu dengan auntukPernyataan: Contoh Simulasi Coin Flip	324
Melakukan Tindakan Beberapa Kali Tertentu dengan auntukPernyataan: Contoh Malam Berbintang	329
Menghitung Melalui Barisan dengan auntukPernyataan: Menghitung Faktorial	332
Bersaranguntukloop	338
BersaranguntukPernyataan dan Piksel: Ringkasan Contoh	341
Grafis	346
Latihan	346
<hr/> Cbab7•Menciptakanruang kosongFungsi	355
ruang kosongFungsi	355
Mode Aktif: Memperkenalkanmempersiapkan()Fungsi	356
Melihat Lebih Dekatmempersiapkan()Fungsi Membuat	358
Sendiriruang kosongContoh Grafis Fungsi dari ruang kosongFungsi Menggunakan Kembali Fungsi	360
	366
	374
Parameter Fungsi dan Argumen Contoh:	378
Menambahkan Parameter ke Fungsi Memberikan	379
Argumen ke Fungsi Menambahkan Beberapa	381
Parameter ke Fungsi Spesial Lainnya	385
ruang kosongFungsi dalam Memproses Suatu Spesialruang kosong	390
Fungsi: Itumenggambar()Fungsi Itumenggambar()	391
Fungsi: Contoh Grafis Menganimasikan Balon dengan menggambar()Fungsi	394
	396
Spesialruang kosongFungsi dalam Pemrosesan Terkait Mouse dan Keyboard	399

Ringkasan	401
Latihan	402
Cbab8■Membuat Fungsi yang Mengembalikan Nilai	409
(Non-ruang kosong)Fungsi	409
Membuat Fungsi Kita Sendiri yang Mengembalikan Nilai	411
Pemanggilan Beberapa Fungsi	418
Menambahkan Parameter ke Fungsi yang Mengembalikan Nilai dengan	420
Melihat Kembali Contoh Segitiga Kanan	427
Beberapa Fungsi dalam Program yang Sama	434
Contoh Lain: Ringkasan Pembulatan ke Tempat	441
Desimal	447
Latihan	448
Cbab9■Array	453
Tentang Array	453
Mendeklarasikan Array	455
Menginisialisasi Array dengan Nilai Tertentu	456
Menampilkan Array: ThecetakArray()Fungsi	458
Mengakses Elemen Array	458
Menyimpan Nilai dalam Array	459
Memproses Array	460
Memilih Kondisi Loop dengan Hati-hati Saat Memproses	464
ArraypanjangProperti Array	465
Memproses Array: Menghitung Suhu Rata-Rata	472
Contoh Grafik: Membuat Grafik Array	480
Mendeklarasikan dan Membuat Array denganbaruContoh	485
Operator Yahtzee: Mengisi Array dengan Angka Acak Nama	488
Variabel Penghitung Loop Lebih Pendek	492
Operator Kenaikan (++) dan Penurunan (--)	493
Contoh Interaktif: Memasukkan Nama	494
Pencarian Array: Pencarian Linier	497
Ringkasan	501
Latihan	502

Cbab10•Pengantar Objek	505
Memahami suatu Objek	505
Mendefinisikan Tipe Objek Baru Dengan Membuat Kelas Baru yang	506
Membuat Instansiasi Objek	509
Mengatur Atribut	511
Menggunakan Metode	512
Mendefinisikan Konstruktor	513
Banyak Objek	518
Berinteraksi dengan	520
Objek Array Objek	527
Ringkasan Pemrograman Berorientasi	529
Objek (OOP).	529
Latihan	530
INDEKS, 533	

Kata pengantar

Pada musim panas 2008, saya diundang untuk memberikan presentasi pada lokakarya CPATH yang disponsori NSF di Universitas La Salle di Philadelphia. Judul lokakaryanya adalah *Revitalisasi Pendidikan Ilmu Komputer melalui Ilmu Media Digital*. Buku pertama saya tentang Pemrosesan baru-baru ini diterbitkan, dan ceramah saya memperkenalkan bahasa dan lingkungan Pemrosesan. Saya juga membahas perjalanan saya sendiri dari pelukis hingga membuat kode. Salah satu peserta lokakarya tampak sangat bersemangat selama saya menyampaikan ceramah, dan antusiasmenya menyebabkan diskusi kelompok menjadi sangat hidup. Ini adalah pertemuan pertama saya dengan Jeff Nyhoff.

Jeff seperti saya adalah seorang hybrid, dengan latar belakang seni (teater) dan komputasi. Jelas sekali, yang membuat Jeff bersemangat selama pembicaraan saya adalah menemukan semangat yang sama; pada saat itu jumlah kita di dunia akademis jauh lebih sedikit (*setidaknya di tempat terbuka*). Jeff dan saya sama-sama terpesona dengan hubungan yang kita lihat antara coding dan praktik seni; persimpangan ini sekarang sering disebut sebagai “pengkodean kreatif”. Daripada melihat dua disiplin ilmu yang sangat berbeda—terkadang digeneralisasikan sebagai aktivitas otak kiri dan kanan yang sangat berlawanan dan bercabang dua—kita melihat integrasi yang indah. Namun kami juga memahami bahwa sebagian besar (dan mungkin jauh lebih waras) orang tidak akan sampai pada kesimpulan ini sendiri. Oleh karena itu, kami berdua berupaya menyebarkan Injil coding yang kreatif, yang telah membawa kita pada misi bersama, dan tujuan akhir dari buku baru yang luar biasa ini: secara radikal mengubah cara komputasi diajarkan.

Saya tidak yakin apakah lokakarya tersebut merupakan paparan awal Jeff terhadap Pemrosesan, namun ia langsung mengenali keanggunan dan kegunaan utamanya, terutama untuk pendidikan pengantar komputasi—sesuatu yang tidak terlalu dipertimbangkan oleh pencetusnya, Ben Fry dan Casey Reas, di luar lokakarya tersebut. konteks seni. Jeff dan saya tetap berhubungan melalui email setelah lokakarya, dan beberapa tahun kemudian, pada konferensi tahunan pendidik ilmu komputer (SIGCSE), kami bertemu kembali secara langsung. Di sanalah, saat berjalan-jalan di pameran penerbit buku SIGCSE, saya bertemu Larry Nyhoff, ayah Jeff. Larry adalah orang yang hangat, ramah, dan rendah hati, dan pada awalnya saya tidak menyadari hubungan profesional Jeff dan Larry selain ayah dan anak. Melalui Jeff, Larry mengenal saya dan pekerjaan saya, dan juga tampak bersemangat dengan kemungkinan yang ditawarkan Pemrosesan di kelas. Saya tidak menyadari bahwa saya sedang berbicara dengan seorang legenda pendidik komputasi sejati.

Mempelajari lebih banyak tentang Larry memberi saya wawasan yang lebih luas tentang bagaimana Jeff, yang mengambil jurusan teater, menjadi profesor ilmu komputer. Larry mengajar ilmu komputer selama lebih dari 40 tahun di Calvin College dan merupakan penulis atau rekan penulis lebih dari 30 buku, yang mencakup berbagai aspek ilmu komputer, serta disiplin ilmu yang terkait (dan tampaknya tidak terkait). Buku-buku Larry mengajarkan berbagai bahasa pemrograman dan keseluruhan upaya penerbitannya

secara harfiah mewakili sejarah pendidikan komputasi modern. Literatur Pemrosesan menjadi lebih terhormat berkat partisipasi Larry.

Jeff dan Larry adalah pendidik yang sangat berkomitmen dan bersedia memberikan pandangan kritis, meskipun penuh kasih sayang, terhadap disiplin mereka sendiri, terutama seputar masalah pedagogi pemrograman. Saya ingat menerima banyak email berapi-api dari Jeff yang membahas kegembiraannya tentang Pemrosesan dan pedagogi Ilmu Komputer, namun juga rasa frustrasinya terhadap keragu-raguan awal komunitas Ilmu Komputer dalam mengadopsi Pemrosesan dan pada akhirnya menerima perubahan. Sungguh menginspirasi bagi saya untuk menemukan seseorang yang benar-benar berkomitmen membantu siswa *Semua murid*, belajar coding. Inilah sebabnya saya sangat bersemangat mengetahui Jeff dan Larry sedang menulis buku mereka sendiri tentang Pemrosesan.

Buku keluarga Nyhoff hadir saat Pemrosesan memasuki bentuk yang sangat stabil dan matang. Hal ini tentunya tidak terjadi pada saya ketika saya mulai buku saya, pada tahun 2004 ketika Processing masih merupakan pekerjaan yang berantakan. Processing 3 sekarang menjadi pustaka kode dan lingkungan pemrograman tingkat profesional yang sangat stabil dan sangat cocok untuk digunakan di dalam kelas Ilmu Komputer dan seterusnya. Namun, meskipun Processing sukses dan kesadaran masyarakat umum semakin meningkat, masih sangat sedikit buku Processing yang dirancang khusus untuk kelas Ilmu Komputer. Warisan Processing adalah komunitas seni dan desain, dan sebagian besar literatur Processing yang ada mendukung populasi ini. Buku *Pengkodean Kreatif dan Seni Generatif 2*, dimana saya adalah salah satu penulisnya, dirancang untuk kelas ilmu komputer, dengan konsentrasi pada contoh visual dan grafis. Pendekatan ini berhasil untuk beberapa populasi, namun tidak semua. Dan di sinilah buku Jeff dan Larry bisa menjadi penyelamat.

Buku baru keluarga Nyhoff secara langsung menargetkan ruang kelas ilmu komputer dengan cara yang tidak dapat dilakukan oleh buku Pemrosesan lainnya. Buku-buku lain, termasuk buku saya, tanpa malu-malu menyimpang dari contoh-contoh berbasis teks tradisional yang ditemukan di hampir semua teks pengantar pemrograman lainnya. Dalam pembelaan kami, memulai perubahan terkadang memerlukan revolusi kecil. Namun, Jeff dan Larry menyajikan pendekatan yang tidak terlalu reaksioner, dengan mengintegrasikan banyak hal menakjubkan tentang Pemrosesan dengan pendekatan tradisional yang telah berhasil dengan baik dalam pedagogi Ilmu Komputer. Pendekatan mereka tidak hanya masuk akal dan efisien, tetapi juga cenderung memberikan kenyamanan lebih bagi instruktur ilmu komputer yang ada (yang mungkin tidak memiliki gelar di bidang teater atau seni lukis).

Ini adalah upaya integrasi yang penuh perhatian—*darisudah dicoba dan benarDanbaru dan ditingkatkan*—yang saya yakini memiliki peluang terbesar untuk memberikan keseimbangan dalam penggunaan Pemrosesan di kelas komputasi.

Ira Greenberg
Dallas, Texas

Kata Pengantar: Mengapa Kami Menulis Buku Ini dan Untuknya Siapa yang Ditulis

Buku ini didasarkan pada keyakinan kami bahwa Pemrosesan adalah bahasa yang sangat baik bagi pemula untuk mempelajari dasar-dasar pemrograman komputer.

Apa itu Pemrosesan?

Pemrosesan awalnya dikembangkan pada tahun 2001 di MIT Media Lab oleh dua mahasiswa pascasarjana, Ben Fry dan Casey Reas, yang ingin mempermudah penggunaan pemrograman komputer untuk menghasilkan seni visual. Fry dan Reas telah bergabung dengan komunitas pengembang yang terus memperbarui dan meningkatkan Pemrosesan. Sebagai proyek perangkat lunak sumber terbuka, Pemrosesan gratis untuk diunduh siapa saja, dan versinya tersedia untuk komputer Windows, MacOS, dan Linux.

Karena Pemrosesan didasarkan pada Java, Pemrosesan cukup bertenaga. Misalnya, jika Anda menjelajahi situs web seperti <https://processing.org> dan <https://www.openprocessing.org>, Anda akan melihat beragam karya yang dibuat menggunakan Pemrosesan: gambar, seni interaktif, simulasi ilmiah, permainan komputer, video musik, visualisasi data, dan banyak lagi. Beberapa dari kreasi ini sangat kompleks.

Karena Pemrosesan berbasis Java, maka memiliki banyak kemiripan dengan Java. Namun, Pemrosesan jauh lebih mudah dipelajari daripada Java. Faktanya, kami percaya bahwa Pemrosesan mungkin adalah solusinya *terbaikbahasa* yang saat ini tersedia untuk mengajarkan dasar-dasar pemrograman komputer kepada pemula.

Mengapa Bukan Java?

Salah satu fitur terpenting dari Pemrosesan adalah memungkinkan kita memulai proses pembelajaran pemrograman dengan membangun program dari pernyataan dan fungsi sederhana. Berbeda dengan bahasa berorientasi objek seperti Java, Pemrosesan tidak mengharuskan kita memulai dengan konsep dan struktur “objek” yang lebih kompleks. Hal ini membuat apa yang kami yakini sebagai pengenalan pemrograman yang lebih lembut. Pada saat yang sama, karena Pemrosesan didasarkan pada Java, kami memiliki kemampuan penuh untuk beralih ke berorientasi objek

pemrograman kapan pun kita siap melakukannya. Faktanya, buku ini mencakup pengenalan dasar tentang objek.*

Pada saat yang sama, karena Pemrosesan didasarkan pada Java, maka biasanya sangat mirip (dan seringkali identik) dengan Java dalam hal sintaksis dan strukturnya, setidaknya dalam kaitannya dengan topik yang biasanya dibahas dalam pengantar pemrograman. kursus. Dengan demikian, kami telah menemukan bahwa siswa melanjutkan dengan sangat lancar dari pemrograman dalam Pemrosesan pada kursus pertama hingga pemrograman dalam Java pada kursus kedua. Secara umum, sintaksis dan struktur Pemrosesan mudah dibawa ke pembelajaran bahasa pemrograman seperti Java, C++, C#, dan JavaScript, jika Anda ingin mempelajari salah satu bahasa pemrograman ini setelah mempelajari Pemrosesan.

Bagaimana dengan Python?

Python menjadi semakin populer sebagai bahasa pemrograman pengantar. Namun, Python sering kali menggunakan sintaksis dan struktur yang agak tidak biasa dibandingkan dengan bahasa pemrograman lain yang mungkin juga ingin dipelajari oleh siswa.

Kami juga menemukan bahwa pengetikan variabel secara dinamis dalam bahasa seperti Python sebenarnya lebih mudah dipahami oleh siswa *setelah* mereka telah mempelajari pengetikan variabel statis dalam bahasa seperti Pemrosesan.[†]

Mengapa Buku Ini?

Ada sejumlah buku bagus tentang Pemrosesan yang telah ditulis. Namun, banyak di antara mereka yang tampaknya pertama-tama dan terutama diarahkan pada penciptaan seni visual. Pemrosesan tentu saja memberikan kemampuan yang kuat untuk melakukan hal ini, namun kecanggihan seni digital yang mengesankan terkadang harus mengorbankan kompleksitas kode yang semakin meningkat, sehingga berpotensi membingungkan programmer pemula.

Dalam buku ini, kami fokus menggunakan Pemrosesan sebagai bahasa untuk mengajarkan dasar-dasar pemrograman kepada pemula yang mungkin memiliki berbagai alasan ingin belajar pemrograman. Oleh karena itu, kami tidak mencoba melakukan survei luas terhadap kemampuan Processing yang luas untuk karya grafis dan interaktif. Sebaliknya, kami menggunakan kemampuannya untuk grafis dan interaktivitas untuk membuat contoh yang kami harap Anda anggap sederhana, ilustratif, menarik, dan bahkan mungkin menyenangkan. Jika tujuan Anda mempelajari program adalah untuk menciptakan seni digital, kami yakin bahwa landasan yang diberikan buku ini adalah tempat yang tepat bagi Anda untuk memulai dan akan mempersiapkan Anda dengan baik untuk buku-buku Pemrosesan lainnya yang lebih berorientasi pada seni digital. Namun, jika Anda ingin belajar pemrograman untuk alasan selain penciptaan seni digital, yakinlah bahwa buku ini juga ditulis untuk Anda.

Mungkin tampak mengejutkan bagi sebagian orang bahwa buku tentang bahasa pemrograman berorientasi grafis seperti Processing memuat banyak contoh yang menggunakan angka atau teks sebagai keluaran utama. Namun, ini digunakan karena berbagai alasan. Pertama, contoh numerik dan tekstual terkadang lebih mudah dipahami dibandingkan contoh grafis. Kedua, sebagian besar

* Bab 10 memperkenalkan objek.

[†]Pemrosesan memang memiliki mode Python, Processing.py, bagi mereka yang ingin belajar memprogram dengan Python Pengolahan. Ada juga P5.js, “interpretasi” JavaScript dari Pemrosesan.

program grafis dalam Pemrosesan memiliki basis numerik yang seringkali lebih mudah dipahami dalam bentuk teks dan keluaran numerik. Ketiga, beberapa pembaca mungkin belajar memprogram dengan tujuan agar mampu memproses informasi numerik (misalnya, jika mereka tertarik pada matematika, sains, atau bisnis) dan/atau memproses data yang mungkin berisi informasi tekstual. Keempat, kami berharap bahwa kombinasi contoh grafis sederhana dengan contoh sederhana yang menggunakan angka dan/atau teks akan menarik bagi peserta didik yang lebih luas. Berdasarkan pengalaman kami, terkadang contoh visual yang pertama kali berhubungan dengan siswa, namun di lain waktu, contoh numerik dan/atau tekstual yang menurut siswa lebih ilustratif. Dalam kasus lain, mungkin saja kombinasi dari contoh-contoh seperti ini yang berhasil menghubungkan siswa.

Kami telah mengikuti apa yang bisa dikatakan adil *tradisional* rangkaian topik, salah satu yang menurut kami berfungsi dengan baik untuk memperkenalkan pemrograman. Selain itu, kami berharap rangkaian seperti itu mungkin terasa familiar bagi instruktur dan siswa yang pernah mempelajari beberapa pemrograman komputer sebelumnya. Kami telah mencoba memperkenalkan konsep-konsep utama ilmu komputer yang terkait dengan pemrograman pengantar tanpa membahasnya secara mendetail sehingga berisiko menjadi berlebihan. Materi pelengkap akan tersedia bagi instruktur yang ingin memperkenalkan lebih banyak lagi konsep ilmu komputer yang terkait dengan topik tersebut (misalnya, dalam kursus "CS1"). Kemampuan pemrosesan untuk animasi dan interaktivitas tidak dieksplorasi secara luas dalam buku ini dan tidak dibahas sampai Bab 7. Namun, contoh tambahan menggunakan animasi dan interaktivitas tersedia bagi instruktur yang ingin memberikan cakupan lebih banyak tentang fitur-fitur ini dan memperkenalkannya lebih awal. Selain itu, beberapa bab online disediakan yang memperkenalkan topik yang sedikit lebih maju dalam Pemrosesan, seperti array dua dimensi, manipulasi string, serta input dan output file (processingnyhoff.com). Latihan tambahan untuk sebagian besar bab juga tersedia.

Sepanjang penulisan teks ini, salah satu perhatian utama kami adalah *ajudari* materi. Beberapa orang mungkin menganggapnya lebih lambat dibandingkan dengan kebanyakan teks pemrograman. Namun, berdasarkan pengalaman kami, mudah untuk melupakan betapa bertahapnya proses yang diperlukan bagi sebagian besar pemula untuk belajar pemrograman. Langkah kami dalam buku ini didasarkan pada pengalaman kami dalam mengajarkan pemrograman kepada berbagai macam pemula di ruang kelas. Tidak diharapkan pengalaman pemrograman sebelumnya. Kami berharap kecepatan membaca buku ini sesuai dengan kecepatan Anda belajar. Kami berharap bahwa teks ini akan berguna bagi siswa dan instruktur di kelas pemrograman pertama, namun kami telah mencoba untuk menulis buku ini sedemikian rupa sehingga juga berguna bagi orang yang belajar pemrograman sendiri.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Ucapan Terima Kasih

Terima kasih banyak kepada Randi Cohen di CRC Press/Taylor & Francis Group atas dukungannya terhadap buku ini dan bimbingan ahlinya selama proses penulisan.

Terima kasih juga kepada Todd Perry di CRC Press/Taylor & Francis Group dan Michelle van Kampen serta rekan-rekannya di Deanta Publishing Services atas pekerjaan produksi mereka dalam penyuntingan, tata letak, dan penyusunan huruf teks ini serta atas saran mereka selama koreksi bacaan kami.

Terima kasih kepada Ira Greenberg atas Kata Pengantaranya dan atas pengaruhnya yang membesarkan hati. Itu adalah salah satu demonstrasi Pemrosesan yang pertama kali meyakinkan kita akan potensi pengajaran yang sangat besar dari bahasa pemrograman ini.

Terima kasih kepada para pengulas yang telah memberikan umpan balik yang sangat membantu dalam menanggapi draf buku ini.

Terima kasih kepada rekan-rekan pengajar di Trinity atas dorongan mereka dan kepada banyak mahasiswa Trinity yang telah menjadi subjek ujian untuk sebagian besar materi dalam buku ini.

Terima kasih kepada keluarga dan teman-teman kami yang telah mendukung kami dalam usaha ini. Syukur kepada Tuhan atas segala nikmatnya, termasuk kesempatan untuk menulis buku ini.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Pendahuluan: Selamat datang di Pemrograman Komputer

Selamat atas keputusan Anda untuk mencoba pemrograman komputer!

Mengapa Belajar Memprogram?

Bisa jadi memprogram komputer adalah sesuatu yang Anda tidak yakin mampu melakukannya atau bahkan ingin melakukannya. Jika ya, maka kami berharap buku ini mengubah pikiran Anda dengan memberikan Anda pengenalan pemrograman komputer yang ramah, lugas, dan solid.

Menjadi “pengguna” komputer berarti menulis program untuk komputer. Namun, kemunculan komputer pribadi yang dimulai pada akhir tahun 1970an dibarengi dengan pertumbuhan ketersediaan perangkat lunak “aplikasi”, seperti pengolah kata dan program spreadsheet. Perangkat lunak tersebut ditulis untuk “pengguna akhir”, yang tidak perlu menjadi pemrogram komputer. Dengan demikian, konsep pengguna komputer sebagai programmer lambat laun bergeser ke konsep seseorang yang menggunakan *perangkat lunak yang ditulis oleh orang lain*.

Namun, masih ada beberapa alasan yang sangat bagus untuk mempelajari pemrograman komputer di era komputasi saat ini.

- **Mempelajari pemrograman komputer mengajarkan Anda tentang apa sebenarnya perangkat lunak itu.**
Program perangkat lunak komputer mungkin tampak rumit dan misterius. Namun, seiring dengan bertambahnya pengalaman Anda dengan contoh perangkat lunak komputer yang lebih sederhana, Anda akan mulai memahami sifat dasar perangkat lunak komputer. Ini adalah pengetahuan yang penting, mengingat semakin banyak bidang kehidupan kita yang dipengaruhi oleh perangkat lunak komputer.
- **Mempelajari pemrograman komputer memberikan wawasan tentang apa yang dilakukan seorang programmer.** Suatu hari nanti, Anda mungkin menemukan diri Anda bekerja dalam tim yang mencakup seorang pemrogram komputer. Mengetahui sesuatu tentang apa yang terlibat dalam pemrograman kemungkinan akan membantu Anda berkomunikasi dan berkolaborasi lebih baik dengan programmer.
- **Daripada mengubah agar sesuai dengan perangkat lunak, Anda dapat merancang cara kerja perangkat lunak.** Ketika kita menggunakan perangkat lunak yang dibuat oleh orang lain, kita sering kali terpaksa menyesuaikan cara berpikir dan bekerja kita dengan cara perangkat lunak tersebut dirancang. Sebaliknya, ketika Anda menulis perangkat lunak Anda sendiri, Anda harus memutuskan apa yang dilakukan perangkat lunak tersebut dan bagaimana cara kerjanya.

- **Anda tidak selalu dapat mengandalkan orang lain untuk membuat perangkat lunak Anda.** Perangkat lunak komputer dibutuhkan oleh banyak orang dan dalam banyak hal sehingga tidak akan pernah ada cukup orang dengan kemampuan pemrograman untuk mengembangkan semua perangkat lunak yang diinginkan pengguna. Oleh karena itu, pengetahuan yang cukup tentang pemrograman komputer untuk membuat atau menyesuaikan perangkat lunak dapat membantu Anda menyediakan perangkat lunak yang Anda atau orang lain butuhkan dan tidak akan tersedia jika tidak.
- **Sedikit pengetahuan tentang pemrograman komputer lebih baik daripada tidak sama sekali.** Banyak orang tidak mempunyai pengalaman sama sekali dengan pemrograman. Oleh karena itu, memiliki sedikit pengetahuan tentang pemrograman komputer akan menempatkan Anda selangkah lebih maju dalam hal pemahaman komputasi.
- **Pemrograman komputer mungkin sudah, atau menjadi, sesuatu yang perlu Anda pelajari.** Banyak bidang studi dan karir yang diperkaya oleh pengetahuan dasar pemrograman komputer, dan semakin banyak yang mulai membutuhkannya. Secara tradisional, pemrograman cocok dipadukan dengan matematika, sains, dan bisnis. Namun di era sekarang, minat dan keterampilan di bidang seni dan media juga sangat cocok dipadukan dengan pemrograman komputer. Saat ini, program komputer digunakan untuk memecahkan berbagai macam masalah dan melayani berbagai tujuan di hampir setiap bidang masyarakat manusia.
- **Pemrograman komputer adalah aktivitas kreatif.** Program komputer sering digunakan untuk memecahkan masalah, namun pemrograman komputer juga merupakan aktivitas kreatif. Oleh karena itu, Anda mungkin mendapati diri Anda menulis sebuah program sebagai sarana untuk mengekspresikan diri atau sekadar karena dorongan untuk menciptakan sesuatu yang baru. Semakin banyak seniman dan desainer yang belajar pemrograman, terutama mereka yang bekerja di media elektronik.
- **Pemrograman komputer mungkin merupakan sesuatu yang Anda kuasai dan sukai.** Anda tidak akan pernah mengetahui hal ini tanpa mencoba pemrograman!

Ini hanyalah beberapa dari banyak alasan bagus untuk belajar pemrograman komputer.

Apa itu Pemrograman?

Pertimbangkan sebuah “program” yang mungkin diberikan kepada Anda di konser band, orkestra, atau paduan suara. Program ini hanya mencantumkan urutan penampilan nomor musik. Gagasan untuk menempatkan sesuatu pada posisi tertentu **urutan** adalah salah satu konsep dasar pemrograman komputer.

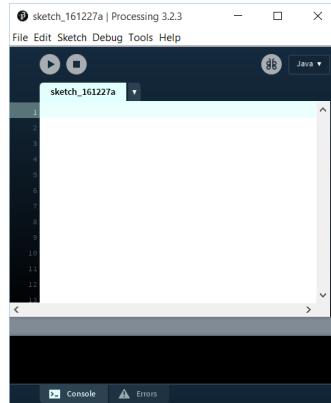
Analogi lain yang berguna adalah resep, di mana tindakan tertentu dengan berbagai bahan dilakukan dalam urutan tertentu untuk mencapai hasil tertentu. Program komputer mirip dengan sebuah resep, yang melibatkan pelaksanaan serangkaian tindakan tertentu dengan menggunakan “bahan” (item informasi) tertentu untuk tujuan menghasilkan hasil tertentu yang diinginkan. Dalam komputasi, resep seperti itu juga dikenal sebagai **algoritma**.

Tur Pemrosesan

Kami akan memberikan pengenalan lebih rinci tentang Pemrosesan di bab pertama, namun mari kita mulai di sini dengan tur singkat tentang Pemrosesan yang akan memberi Anda sedikit gambaran tentang seperti apa dan ke mana tujuan kita dalam buku ini. *Jangan khawatir* jika Anda belum memahami semua detail yang disajikan dalam tur ini; segala sesuatu dalam pendahuluan ini akan dijelaskan lebih lengkap pada bab-bab selanjutnya.

Pengolahan adalah *sumber terbuka* proyek. Itu dapat diunduh secara gratis dari <https://processing.org> dan tersedia untuk komputer Windows, Mac, dan Linux.*

Saat Anda membuka Pemrosesan, yang Anda lihat adalah lingkungan pengembangan terintegrasi atau IDE Pemrosesan:

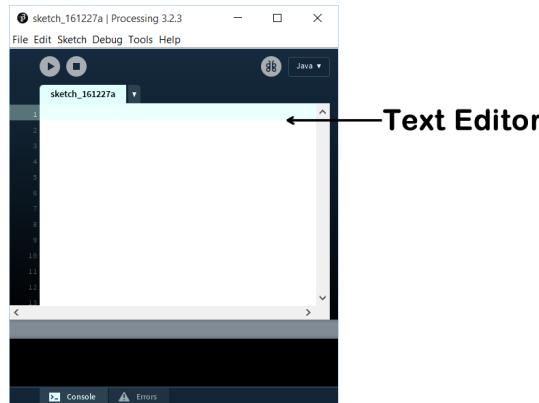


IDE hanyalah sebuah lingkungan perangkat lunak tempat Anda dapat mengembangkan dan menguji program. Dibandingkan dengan kebanyakan IDE, IDE Pemrosesan jauh lebih sederhana untuk digunakan. Secara resmi, IDE Pemrosesan dikenal sebagai "Lingkungan pengembangan pemrosesan" atau PDE. Di sini, di PDE, Anda dapat memasukkan program dan meminta Pemrosesan menjalankannya.

"Halo Dunia!" Contoh

Merupakan tradisi dalam pemrograman komputer untuk memulai dengan sebuah program yang hanya menampilkan kata-kata "Halo, Dunia!" Mari kita buat program seperti itu di Processing.

Di bagian Editor Teks pada jendela Pemrosesan,

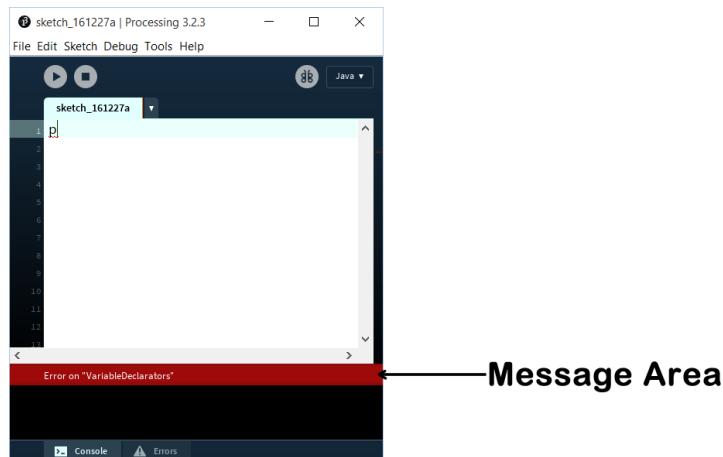


*Saat tulisan ini dibuat, rilis resmi terbaru adalah versi 3.3.

masukkan dengan hati-hati yang berikut ini:

```
print("Halo Dunia!");
```

Untuk saat ini, saat Anda mengetik, abaikan pesan kesalahan apa pun yang muncul pada bilah merah di Area Pesan pada jendela Pemrosesan:



Setelah Anda selesai mengetik, pesan kesalahan ini akan hilang. Bagian atas area Editor Teks pada jendela Pemrosesan sekarang akan menyerupai berikut ini:

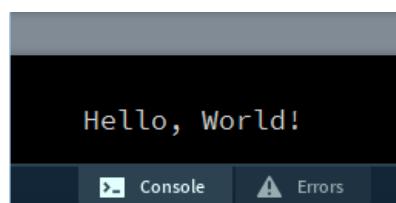
```
1 print("Hello, World!");
```

Anda sekarang telah menulis satu baris **kodedalam Pengolahan**. Baris kode khusus ini juga merupakan contoh dari apa yang disebut **pernyataan**, karena merupakan instruksi lengkap untuk melakukan tindakan tertentu. Saat kita menjalankan program ini, Pemrosesan akan menjalankan instruksi ini dengan melakukan tindakan yang dijelaskannya.

Selanjutnya, klik **Berlari** (tombol, yang terdapat di area jendela Pemrosesan yang dikenal sebagai toolbar:



Anda akan melihat "Halo, Dunia!" ditampilkan di bagian hitam di bagian bawah jendela Pemrosesan. Bagian dari jendela Pemrosesan ini dikenal sebagai **menghibur**.



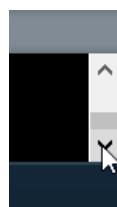
Jika Anda tidak melihat hal di atas ditampilkan di konsol, berikut beberapa hal yang perlu diperiksa di kode Anda:

- Kata mencetak harus ditulis seluruhnya dengan huruf kecil semua.
- Pastikan Anda memiliki tanda kurung kiri dan kanan.
- Pesan yang akan ditampilkan, "Halo, Dunia!" harus diapit oleh sepasang tanda kutip ganda.
- Pernyataan yang Anda masukkan harus diakhiri dengan titik koma.

Setelah Anda melihat pesan "Halo, Dunia!" salam, Anda telah berhasil menulis dan menjalankan program pertama Anda! Anda sekarang dapat menghentikan program dengan menekan **Berhenti** tombol:



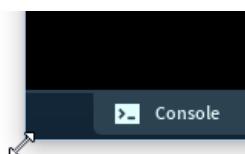
Perhatikan bilah gulir di tepi kanan konsol:



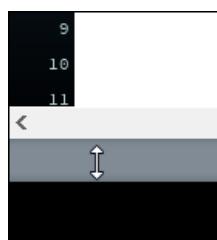
Bilah ini memungkinkan Anda menelusuri baris keluaran konsol.

Ada kalanya berguna untuk memperbesar area konsol untuk melihat lebih banyak keluaran program ini. Untuk melakukan ini:

- 1) Klik dan seret di tepi jendela Pemrosesan untuk memperbesar jendela ini.



- 2) Klik dan seret ke atas pada Bilah Pesan untuk memperbesar konsol.

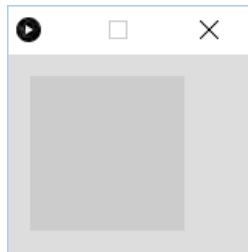


Anda sekarang dapat melihat lebih banyak baris keluaran konsol tanpa harus menggulir.

Sebuah Pemrosesan "Sketsa"

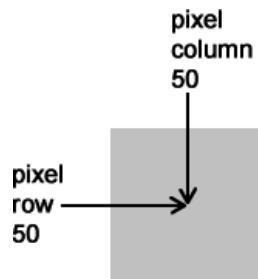
Menampilkan informasi di konsol, seperti "Halo, Dunia!" salam, bisa sangat berguna saat pemrograman. Namun, ada alasan mengapa program dalam Pemrosesan juga disebut "sketsa": pemrograman dalam Pemrosesan juga dapat melibatkan *menggambar*.

Seperti yang mungkin Anda ketahui, ketika Anda mengklik tombol Jalankan, Pemrosesan juga membuka apa yang disebut **terminal**. Di dalam jendela ini terdapat area abu-abu persegi panjang yang akan kita sebut sebagai **kanvas** karena dapat diambil menggunakan fungsi Pemrosesan.



Secara default, sketsa Pemrosesan akan merender (menghasilkan) kanvas yang terdiri dari 100 kolom piksel dan 100 baris piksel. **Piksel** adalah kotak kecil dengan satu warna. Dalam hal ini, semua piksel yang membentuk kanvas berwarna abu-abu.

Kolom piksel diberi nomor dari kiri ke kanan, dimulai dengan nol. Demikian pula deretan piksel diberi nomor dari atas ke bawah, dimulai dari nol. Dengan demikian, setiap piksel mempunyai lokasi yang unik, dijelaskan oleh kolom dan baris di mana ia berada. Misalnya, piksel di kolom 50 dan baris 50 berada di dekat bagian tengah kanvas saat ini.



Menggambar Elips

Mari kita hapus pernyataan kita saat ini yang menginstruksikan Pemrosesan untuk menampilkan "Halo, Dunia!" di konsol. Area Editor Teks pada jendela Pemrosesan sekarang seharusnya kosong:



Sekarang, masukkan yang berikut ini ke dalam area Editor Teks di jendela Pemrosesan:

```
elips(50, 50, 20, 10);
```

Ini adalah pernyataan yang menginstruksikan Processing untuk menggambar elips. Ini mencakup empat item informasi yang diperlukan untuk menentukan elips yang akan digambar.

Item informasi pertama dan kedua menentukan di mana *tengah* elips masing-masing harus berupa: kolom piksel dan baris piksel. Dalam pernyataan saat ini, kami telah memilih lokasi piksel kolom 50 dan baris 50, di tengah kanvas:

```
elips(50,50, 20, 10);
```

Item ketiga dan keempat menentukan *ukuran* elips yang akan digambar: masing-masing lebar elips dalam piksel dan tinggi elips dalam piksel. Dalam pernyataan saat ini, kami telah memilih elips dengan lebar 20 piksel dan tinggi 10 piksel:

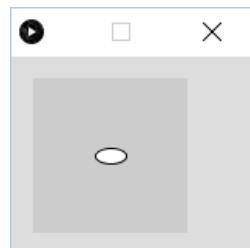
```
elips(50, 50, 20, 10);
```

Perhatikan bahwa kami telah menambahkan *aruang angka* setelah setiap koma. Spasi ini tidak diperlukan, namun "spasi" semacam ini dapat membuat kode lebih mudah dibaca. Dalam kebanyakan kasus, Pemrosesan sangat memaafkan penambahan spasi ekstra dan baris kosong untuk tujuan ini.

tekan **Berlari** tombol.



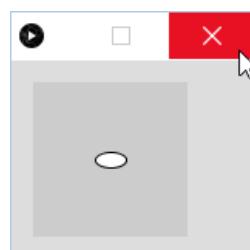
Anda sekarang akan melihat elips dengan pusatnya terletak di kolom piksel 50 dan baris piksel 50, dengan lebar 20 piksel dan tinggi 10 piksel:



Anda sekarang dapat menghentikan program dengan menekan **Berhenti** tombol:



Alternatifnya, Anda dapat mengklik untuk menutup jendela tampilan:



Animasi dan Interaktivitas

Program menggambar elips sebelumnya memberi Anda demonstrasi singkat tentang jenis program yang akan kami minta untuk dijalankan oleh Pemrosesan. Ini adalah contoh program yang ditulis dengan apa yang disebut**modus statis**. Kita akan menggunakan mode statis untuk sebagian besar buku ini karena memungkinkan kita menulis program yang lebih sederhana.

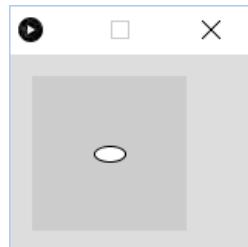
Namun, salah satu hal yang membuat Processing kuat dan populer adalah kemampuannya untuk membuat gambar animasi dan interaktif. Kemampuan yang lebih maju ini tidak akan dieksplorasi secara luas dalam buku ini dan tidak akan diperkenalkan sampai Bab 7 karena kemampuan tersebut mengharuskan kita untuk melakukan pemrograman yang lebih maju menggunakan Pemrosesan.**modus aktif**Danmenggambar()fungsi. Namun, sementara ini, berikut ini hanyalah sedikit contoh bagaimana hal ini dapat dilakukan dalam Pemrosesan. Lagi,*jangan khawatir*jika Anda tidak memahami semua yang kami lakukan. Untuk saat ini, cukup ketik kodennya dan lihat apa yang terjadi.

Mulailah dengan memodifikasi program Anda agar sesuai dengan hal berikut:

```
batalkan undian()
{
    elips(50, 50, 20, 10);
}
```

Apa yang kami lakukan di sini adalah memasukkan program kami saat ini ke dalam apa yang disebut menggambar()fungsi. Apa yang istimewa dari itumenggambar()fungsi adalah secara otomatis melakukan “loop”. Dengan kata lain, Processing secara otomatis akan melakukan pernyataan apa pun yang terdapat di dalamnya menggambar()berfungsi berulang kali ketika kita menjalankan program kita.

Saat kita menekan**Berlaritombol**, kami tidak melihat perubahan pada sketsa kami:



Hal ini karena Processing adalah menggambar elips dengan ukuran yang sama pada posisi kanvas yang sama, berulang-ulang.

tekan**Berhentitombol**.

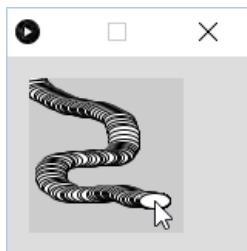
Selanjutnya, mari kita ubah pernyataan yang telah kita tempatkan di dalammenggambar()berfungsi sebagai berikut:

```
batalkan undian()
{
    elips(tikusX,tikusY, 20, 10);
}
```

Di sini, kami telah mengganti angka spesifik yang menjelaskan kolom piksel dan baris piksel untuk lokasi pusat elips dengan **tikusX**DantikusY. (Perhatikan keduanya

dari entri ini diakhiri dengan huruf kapital:**mouseX**Dan**mouseY**.) Ini adalah dua “variabel” yang dibangun dalam Pemrosesan dan terus melacak lokasi piksel penunjuk tetikus saat ini ketika berada di atas kanvas. (Kita akan belajar lebih banyak tentang variabel di bab-bab berikutnya.)

Sekarang, tekan **Berlari**tombol. Jika Anda mengarahkan penunjuk tetikus ke atas kanvas, Anda akan melihat sesuatu seperti berikut ini muncul:



Kami melihat hasil ini karena, berulang kali, Pemrosesan menggambar elips berukuran 20 piksel kali 10 piksel di lokasi penunjuk tetikus saat ini. Anda sekarang memiliki *interaktif*/Memproses sketsa.

Semoga tur singkat ini memberi Anda gambaran tentang sifat Pemrosesan. Sekali lagi, sketsa animasi dan interaktif terakhir ini memanfaatkan beberapa fitur lanjutan yang tidak akan dibahas hingga Bab 7, jadi jangan khawatir jika tidak memahami semua detail dalam contoh ini. Yakinlah, fitur Pemrosesan akan diperkenalkan kepada Anda secara menyeluruh dan bertahap sepanjang buku ini.

Latihan

- 1) Ubah pesan “Halo, Dunia!” contoh dari bab ini sehingga ketika Anda menjalankan program, nama depan Anda ditampilkan di salam.

Contoh:Halo, Rebecca!

- 2) Ubah kode dari menu “Halo, Dunia!” contoh dalam bab ini sebagai berikut:

```
cetakln("Halo Dunia"); cetakln("Selamat  
datang di Pemrosesan!");
```

Jalankan program yang dimodifikasi untuk melihat keluaran apa yang dihasilkan.

- 3) Ubah contoh sebelumnya sehingga menghasilkan keluaran sebagai berikut:

Halo Dunia!
Selamat datang di Pemrosesan!
Mari kita mulai.

4) Lakukan perubahan berikut pada contoh program interaktif dari bab ini:

```
batalkan undian()
{
    elips(mouseX, mouseY,10,20);
}
```

Jalankan program ini dan jelaskan perubahan hasilnya.

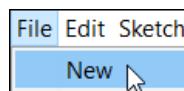
Gambar Dasar dalam Pengolahan

Sekarang kita akan mengesampingkan kemampuan tingkat lanjut yang terkait dengan perulangan menggambar(fungsi dan Pemrosesan) yang kita jelajahi di akhir pendahuluan. Kami akan kembali ke Pemrosesan yang lebih sederhana(modus statis) dan kerjakan beberapa contoh sederhana gambar dasar dalam Pemrosesan. Contoh-contoh ini juga akan memberi kita kesempatan untuk melihat lebih dekat beberapa dasar-dasar pemrograman komputer secara umum.

Bab ini bukanlah pengenalan lengkap tentang kemampuan menggambar Processing yang luas. Sebaliknya, tujuannya adalah memberi Anda dasar yang cukup dalam menggambar dengan Processing untuk memahami contoh grafis di bab-bab selanjutnya.

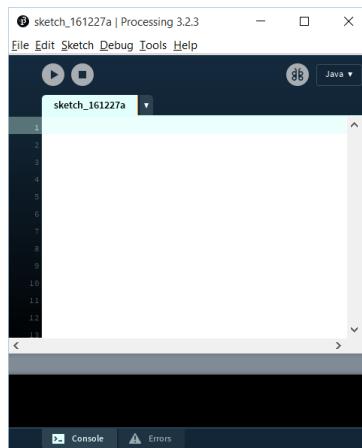
Memulai Program Baru

Di jendela Pemrosesan, tarik ke bawah **Mengajukan** menu dan pilih **Baru**.



Mulai sekarang kami akan menjelaskan pilihan menu seperti **Mengajukan>Baru**.

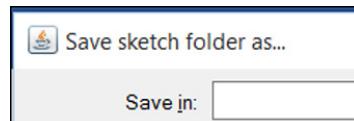
Kita sekarang melihat jendela sketsa Pemrosesan baru:



2■Pemrosesan: Pengantar Pemrograman

Menyimpan Program

Saat memulai program baru, ada baiknya untuk segera menyimpannya dan sering menyimpannya setelah melakukan perubahan pada program. "Menabung lebih awal dan sering!" adalah bagaimana strategi ini kadang-kadang dinyatakan. Ini adalah nasihat bagus yang dapat membantu menghindarkan Anda dari kehilangan pekerjaan yang telah Anda lakukan. Menyimpan program di Processing hampir sama dengan menyimpan pekerjaan Anda di perangkat lunak lain yang Anda gunakan. Pilih **Mengajukan>Menyimpan**. Karena ini adalah pertama kalinya Anda menyimpan program ini, maka "Simpan folder sketsa sebagai..." kotak dialog muncul.

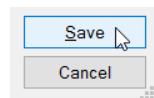


Dengan menggunakan dialog ini, mulailah dengan memilih lokasi folder file tempat Anda ingin menyimpan program Pemrosesan yang sedang Anda tulis. Secara default, Pemrosesan akan menyimpan semua sketsa (program) Anda di lokasi file yang dikenal sebagai milik Anda **Buku skets**. Pengaturan program Pemrosesan yang menentukan lokasi file Buku Sketsa default ini dapat diubah jika Anda berencana untuk menyimpan program Pemrosesan di lokasi lain.* Anda juga dapat menelusuri lokasi folder file yang diinginkan seperti yang biasa Anda lakukan saat menyimpan pekerjaan Anda di program perangkat lunak lain yang Anda gunakan.

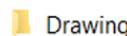
Selanjutnya, di **Nama file** kotak teks di dalam kotak dialog, masukkan nama untuk program Anda. Merupakan kebiasaan untuk menggunakan huruf kapital pada huruf pertama program Pemrosesan. Mari kita beri nama contoh ini **Menggambar**.



Kemudian, klik pada **Menyimpan** tombol untuk menyelesaikan penyimpanan program ini.



Hasilnya, Pemrosesan menciptakan yang baru **map** di mana program Anda disimpan. Folder ini memiliki nama yang sama dengan yang Anda berikan pada program Anda. Jadi, dalam contoh ini, folder baru diberi nama **Menggambar**.



Pemrosesan menyebutnya sebagai **folder sketsa**. Jika Anda memeriksa isi folder sketsa, Anda akan melihat file yang memiliki nama yang sama dengan folder tersebut tetapi juga memiliki **.pde** ekstensi nama file. Misalnya, di folder sketsa saat ini, **Menggambar.pde** adalah nama file yang berisi file kita **Menggambar** program.

* Anda dapat mengubah lokasi default Buku Sketsa ini dengan melakukan hal berikut:

Pilih **File > Preferensi**.

Klik **Jelajah** hitombol di sebelah kotak teks berlabel **Lokasi Buku Sketsa**. Telusuri ke folder tempat Anda ingin menyimpan program Pemrosesan Anda. tekan **Buka** katombol.

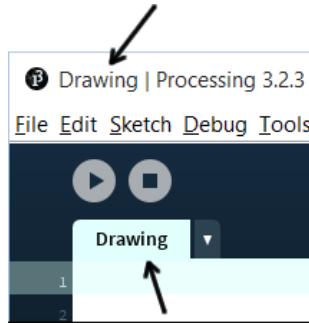
tekan **OKE** tombo. Folder yang Anda pilih sekarang menjadi lokasi default tempat Pemrosesan akan menyimpan dan mengambil program Anda.

Drawing.pde

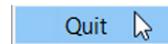
Jika kamu melakukan *bukanlihat.pde* bagian dari nama file ini, maka Anda perlu mengaktifkan tampilan ekstensi nama file di pengaturan sistem operasi komputer Anda jika Anda ingin melihat ekstensi nama file tersebut.*

Setiap .pdeFile yang dibuat oleh Processing hanyalah file teks yang menyimpan semua baris kode yang kita tulis untuk program tertentu.

Perhatikan bahwa setelah kita menyimpan program kita, nama yang kita berikan untuk program ini sekarang muncul di bagian atas jendela Processing dan pada tab tepat di atas area Editor Teks:



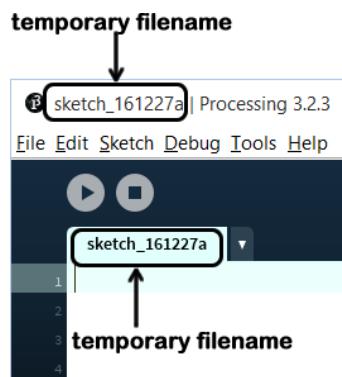
Untuk saat ini, tutup Pemrosesan dengan memilih **File > Keluar**.



Mengambil Program

Mulai Ulang Pemrosesan.

Perhatikan bahwa Pemrosesan selalu dimulai dengan *a baru* program. Pemrosesan menunjukkan bahwa program baru ini belum disimpan dengan nama dengan menampilkan nama sementara (berdasarkan tanggal saat ini) di jendela Pemrosesan dan tab tepat di atas area pengeditan:

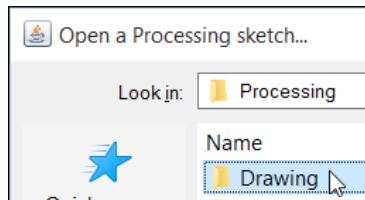


Berikut cara yang dapat diandalkan untuk mengambil kembali program yang telah Anda tulis sebelumnya dan simpan di Pemrosesan. Pilih **File > Buka**. Pada dialog yang muncul, cari **Menggambarmap**. (Jika

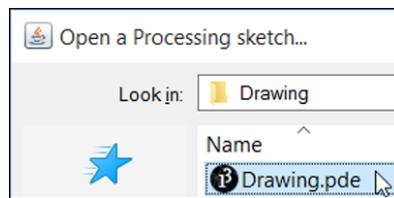
* Misalnya, di Windows 10, cukup buka File Explorer, klik tab View, dan centang kotak berlabel "File name extensions." Untuk Mac yang menjalankan OS X, buka Finder > Preferences lalu pilih "Show all filename extensions."

4■Pemrosesan: Pengantar Pemrograman

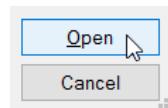
Anda menyimpan program Anda di lokasi selain lokasi default, telusuri ke lokasi lain tersebut.)



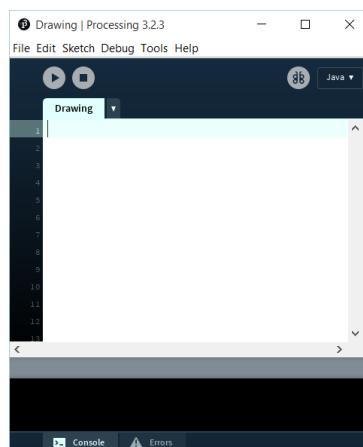
Klik dua kali Menggambar folder untuk membukanya. Anda sekarang akan melihat file bernama **Menggambar.pde** yang berisi program Anda. (Jika Anda tidak mengaktifkan ekstensi file, cukup Menggambar akan ditampilkan sebagai nama file ini.) Pilih file ini.



Selanjutnya, tekan **Membuka** tombol.



Anda sekarang harus melihat milik Anda Menggambar program terbuka sekali lagi di Pemrosesan:



Memasukkan Kode ke dalam Editor Teks

Kita Menggambar program belum berisi baris kode apa pun:

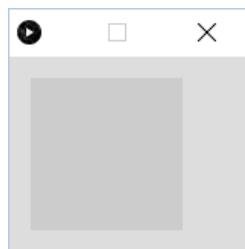


Namun, Processing tetap memungkinkan kita menjalankan program ini. Kita dapat menjalankan suatu program dengan menekan tombol **Berlari** tombol.



Alternatifnya, kita bisa memilih **File > Jalankan.***

Ketika kita menjalankan program menggambar, kita sekali lagi melihat kanvas default yang dihasilkan oleh Processing di dalam jendela tampilan:



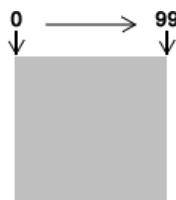
Kanvas ini adalah area persegi panjang yang, secara default, terdiri dari kolom dan baris piksel abu-abu.

Katapike adalah kependekan dari “elemen gambar”. Istilah ini mencerminkan fakta bahwa piksel hanyalah sebuah kotak berwarna tunggal yang merupakan satu bagian dari gambar yang lebih besar.[†]

Seperti yang kita pelajari sebelumnya, **lebar bawaan** dari kanvas Pemrosesan adalah **100** kolom piksel:



(*Piksel*terkadang disingkat *piksel*.) Kolom piksel ini diberi nomor dari kiri ke kanan, dimulai dari 0. Jadi, untuk kanvas default, nomor kolom piksel dimulai dari 0 hingga 99:

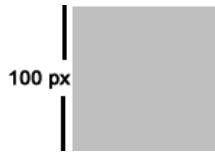


Beginu pula dengan **tinggi standard** dari kanvas adalah **100** deretan piksel.

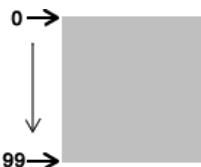
* Menu file juga menampilkan pintasan keyboard untuk menjalankan program: Ctrl-R untuk Windows, Command-R (⌘-R) untuk Mac.

[†]Ini bisa berupa potongan gambar di layar, gambar yang dicetak, atau gambar dalam file. Syarat *piksel* digunakan dalam semua ini konteksnya, yang terkadang membuatnya menjadi istilah yang rumit.

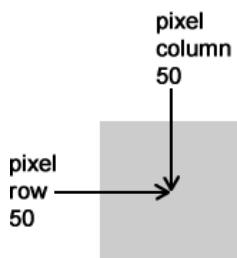
6■Pemrosesan: Pengantar Pemrograman



Baris piksel ini diberi nomor dari atas ke bawah, dimulai dari 0. Jadi, untuk kanvas default, nomor baris piksel juga dimulai dari 0 hingga 99:



Oleh karena itu, setiap piksel dapat dikatakan memiliki lokasi unik, yang dijelaskan oleh kolom dan baris di mana piksel tersebut ditemukan. Misalnya, piksel di dekat bagian tengah kanvas default adalah piksel pada kolom 50 dan baris 50:



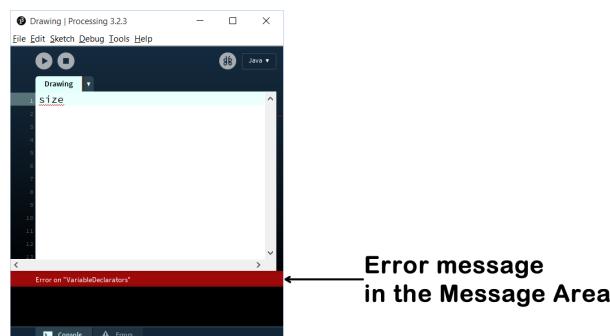
Pemrosesan memiliki banyak bawaan**fungsi** yang tersedia untuk kita gunakan. Misalnya, selama “tur” Pemrosesan, kami menggunakan dua fungsi berikut:

mencetak()
menampilkan teks di konsol
elips()
menggambar elips di kanvas

Selain itu, **ukuran()** fungsi memungkinkan kita mengatur ukuran kanvas yang kita gunakan. Misalnya, masukkan pernyataan berikut ke dalam program kita:

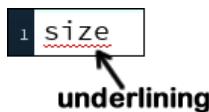
ukuran (150, 200);

Perhatikan bahwa saat Anda mengetik, area jendela Pemrosesan yang dikenal sebagai tampilan Area Pesan berubah menjadi merah dan menampilkan pesan kesalahan:

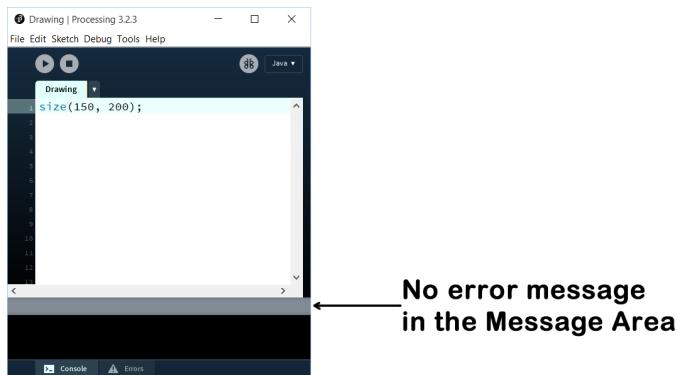


Jangan khawatir dengan ini!

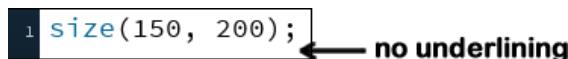
Selain itu, saat Anda mengetik, bagian tertentu dari apa yang Anda masukkan mungkin digarisbawahi dengan garis berlekuk-lekuk berwarna merah. Sekali lagi, jangan khawatir dengan hal ini.



Setelah Anda selesai mengetik baris kode, pesan kesalahan akan hilang dari Area Pesan:



Selain itu, garis berlekuk-lekuk merah akan hilang dari bawah baris kode Anda:



Pesan kesalahan dan garis bawah oleh Processing mungkin sedikit membingungkan pada awalnya. Namun, saat Anda mulai belajar memprogram, sebaiknya abaikan saja peringatan dari Pemrosesan ini hingga Anda selesai sepenuhnya masukkbaris kode Anda. Jika Anda telah memasukkan baris kode dengan benar, pesan peringatan dan garis berlekuk-lekuk akan hilang setelah Anda selesai memasukkannya.

Setelah Anda terbiasa dengan peringatan ini, Anda akan menyadari bahwa peringatan tersebut sebenarnya cukup berguna. Misalnya, coba hapus titik koma di akhir baris kode yang baru saja Anda masukkan:

size (150, 200)
↑
delete the semicolon

Perhatikan bahwa, sekarang, garis berlekuk-lekuk merah telah muncul di bawah tempat di baris kode tempat kita sebelumnya memasukkan titik koma:

size (150, 200)~~
↑

Perhatikan juga bahwa pesan kesalahan sekali lagi muncul di Area Pesan pada jendela Pemrosesan:

Missing a semicolon ";"

8■Pemrosesan: Pengantar Pemrograman

(Catatan: Jika Anda tidak melihat pesan kesalahan ini di Area Pesan, maka klik pada garis merah berlekuk-lekuk dengan mouse Anda, dan pesan kesalahan akan muncul.) Ini adalah pesan kesalahan yang berguna. Sebagian besar pernyataan dalam Processing harus diakhiri dengan titik koma, dan Processing mengingatkan kita di sini bahwa titik koma memang diperlukan dalam kasus ini. Pemrosesan juga memperingatkan kita bahwa jika kita mencoba menjalankan program dalam kondisi saat ini, kita akan menerima kesalahan. Sebagian besar pesan kesalahan Pemrosesan memberi kita informasi berguna seperti ini.

Saat Anda memprogram dengan Pemrosesan, terkadang Anda akan menemukan pesan kesalahan yang tidak sepenuhnya jelas. Namun, salah satu peningkatan besar dalam versi terbaru Pemrosesan adalah bahwa pesan kesalahan menjadi lebih mudah dipahami oleh pemrogram pemula.

Segera setelah kami memperbaiki kesalahan kami dengan memasukkan kembali titik koma

```
size(150, 200);  
↑  
add the semicolon
```

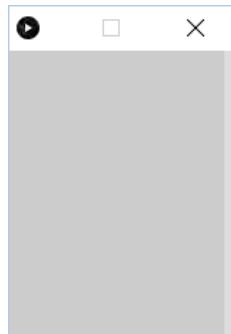
garis berlekuk-lekuk sekali lagi menghilang, dan pesan kesalahan menghilang dari Area Pesan.

Simpan ini Menggambar program dengan memilih **Mengajukan>Menyimpan**. Kami akan terus mengerjakan program ini di bagian selanjutnya.

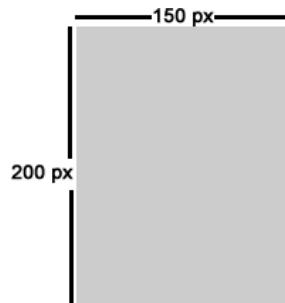
Menggambar Dasar dengan Elemen Grafis

Mengatur Ukuran “Kanvas”: Melihat Lebih Dekatukuran()Fungsi

Saat kita menjalankan milik kita Menggambar program lagi, sekarang kita melihat tampilan jendela berikut:



Di dalam jendela tampilan terdapat kanvas dengan lebar kolom 150 piksel dan tinggi baris 200 piksel.



Mari kita lihat lebih dekat penggunaan kamiukuran()fungsi. Saat kita menggunakan fungsi dalam pernyataan seperti ini, kita mengatakan bahwa kita panggilanfungsi ini. Seperti kebanyakan fungsi, ukuran()fungsi memerlukan beberapa informasi dari kami agar dapat beroperasi saat kami memanggilnya. Kami menempatkan informasi yang diperlukan tersebut di dalam tanda kurung setelah nama fungsinya. Informasi yang diperlukan seperti itu disebut **parameter**. Barang yang kami sediakan untuk memenuhi persyaratan ini dikenal sebagai **argumen**. Dengan kata lain, untuk masing-masing diperlukan *parameter*, kita perlu menyediakan yang sesuai *argumen*.

Ituukuran()fungsi memiliki *dua*diperlukan *parameter*, jadi kita perlu menyediakan *dua*sesuai *argumen*: piksel yang diinginkan/*lebar* dan piksel yang diinginkan *tinggi*, tercantum dalam urutan itu, dipisahkan dengan koma.

Jadi, bentuk dasar panggilan keukuran()fungsinya dapat dijelaskan seperti ini:

ukuran(*lebar,tinggi*);

Kata-kata yang dicetak miring adalah parameter yang diperlukan:

<i>lebar</i>	Lebar piksel kanvas yang diinginkan
<i>tinggi</i>	Tinggi piksel kanvas yang diinginkan

Katasintaksiskadang-kadang digunakan dalam pemrograman komputer untuk menggambarkan urutan dan struktur yang benar dari komponen tertentu yang membentuk elemen yang lebih besar dari bahasa pemrograman. Misalnya bentuk dasar panggilan keukuran()Fungsi juga dikenal sebagai sintaks fungsi itu.

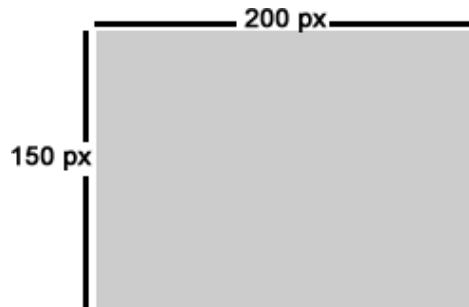
Dalam program kami saat ini, kami telah memberikan argumen berikut dalam panggilan kami keukuran() fungsi, menentukan kanvas yang diinginkan/*lebar* dan kanvast*tinggi*, masing-masing:

ukuran(150,200);

Penting untuk diperhatikan bahwamemesandi mana kami memberikan argumen-argumen ini sangatlah penting. Misalnya, jika kita membalik angkanya

ukuran(200,150);

lalu kanvas yang ditentukan/*lebar*akan menjadi 200 kolom piksel, dan kanvas yang ditentukant*tinggi*akan menjadi 150 baris piksel. Dua argumen untukukuran()fungsi akan menentukan kanvas dengan ukuran berbeda:



10■Pemrosesan: Pengantar Pemrograman

Jadi, pastikan argumen dalam panggilan Anda sesuai ukuran() fungsi muncul dalam urutan yang benar:

```
ukuran(150,200);
```

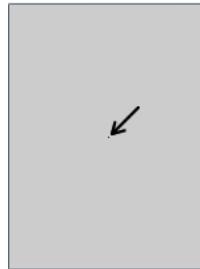
Pilih **File > Simpan** untuk menyimpan kembali program saat ini. Kami akan terus mengerjakan program ini di bagian berikutnya.

Poin Gambar: Titiktitik()Fungsi

Kita dapat menggambar “titik” pada lokasi piksel tertentu menggunakan titik() fungsi. Misalnya, mari tambahkan pernyataan berikut ke program kita:

```
ukuran (150, 200);  
titik(75, 100);
```

Saat kita menjalankan program dan melihat dengan cermat, kita melihat satu piksel hitam di dekat bagian tengah kanvas:



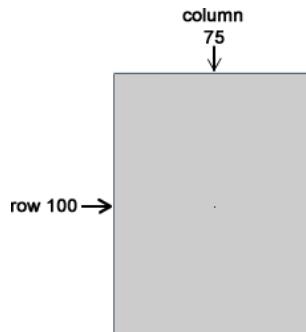
Seperti yang telah kita lihat, secara default, Processing menggambar sebuah titik di kanvas sebagai satu piksel hitam. Mengapa titik imbang di lokasi kanvas tertentu? Karena argumen pertama yang kami berikan kepada titik() fungsi

```
titik(75, 100);
```

menentukan piksel *kolom* dimana titik tersebut akan ditempatkan. Argumen kedua yang kami berikan kepada titik() fungsi

```
titik(75,100);
```

menentukan piksel *baris* dimana titik tersebut akan ditempatkan.



Ringkasnya, bentuk dasar (sintaks) suatu pernyataan memanggil ketitik()fungsinya adalah
titik(*kolom,baris*);

Parameternya adalah

<i>kolom</i>	Kolom piksel tempat titik akan digambar
<i>baris</i>	Baris piksel tempat titik akan digambar

Program kami saat ini terdiri dari dua pernyataan:

ukuran (150, 200);
titik(75, 100);

Sangat penting untuk dipahami bahwa ketika kita menjalankan program kita, pernyataan-pernyataan ini dieksekusi (dilakukan)*dengan Memproses sesuai urutan yang terdaftar.*

- 1)ukuran (150, 200); Menghasilkan kanvas kosong berukuran 150 piksel kali 200 piksel
- 2)titik(75, 100); Menarik sebuah titik pada kolom piksel 75, baris piksel 100

Misalnya, jika kita mengubah urutan pernyataan ini menjadi

titik(75, 100);
ukuran (150, 200);

maka urutan eksekusinya adalah

- 1)titik(75, 100); Menggambar titik pada kolom piksel 75, baris piksel 100 pada *bawaan* Kanvas 100 piksel kali 100 piksel
- 2)ukuran (150, 200); Menghasilkan kanvas kosong berukuran 150 piksel kali 200 piksel,*menghapus* titik yang ditarik

Alhasil, saat kita menjalankan program, kita akan melihat kanvas kosong.

Jadi, pastikan dua pernyataan dalam program Anda muncul dalam urutan berikut:

ukuran (150, 200);
titik(75, 100);

Pilih **File > Simpan**untuk menyimpan kembali program ini. Kami akan terus mengerjakannya di bagian berikutnya.

Segmen Garis Gambar: Thegaris()Fungsi

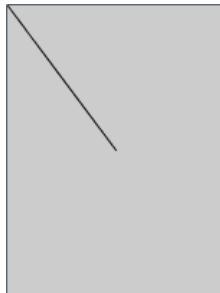
Kita juga dapat menggambar segmen garis di Processing menggunakan **garis()**fungsi. Sebagai contoh, mari kita ubah pernyataan kedua dalam program kita menjadi berikut:

ukuran (150, 200);

12■Pemrosesan: Pengantar Pemrograman

baris(0, 0, 75, 100);

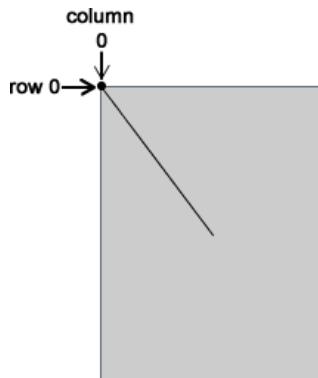
Sekarang, ketika kita menjalankan program kita, kita melihat yang berikut:



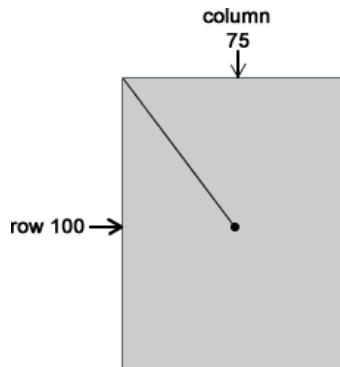
Ruas garis khusus ini ditarik karena *Pertama* dan *Kedua* argumen yang kami berikan kepada garis() fungsi

garis(0, 0, 75, 100);

tentukan kolom dan baris masing-masing lokasi piksel dari salah satu titik akhir segmen garis:



Beginu pula dengan *ketiga* dan *keempat* argumen yang kami berikan kepada garis() fungsi menentukan kolom dan baris, masing-masing, dari lokasi piksel *lainnya* titik akhir ruas garis:



Singkatnya, bentuk dasar panggilan kegaris()fungsinya adalah

baris (kolom₁, baris₁, kolom₂, baris₂);

Parameternya adalah

- kolom₁** Lokasi kolom piksel pada titik akhir pertama
- baris₁** Lokasi baris piksel pada titik akhir pertama
- kolom₂** Lokasi kolom piksel pada titik akhir kedua Lokasi
- baris₂** baris piksel pada titik akhir kedua

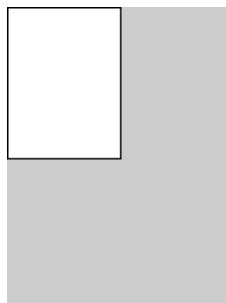
Jadi, kami cukup menyediakan lokasi piksel dari titik akhir segmen garis dan Pemrosesan akan melakukan sisanya. Ini menentukan piksel mana yang perlu diwarnai ulang untuk menghasilkan segmen garis yang kita tentukan dalam argumen panggilan kita kegaris()fungsi. (Secara default, ini adalah segmen garis berwarna hitam dan tebalnya satu piksel.) Proses di mana Pemrosesan mengetahui warna yang dibutuhkan untuk setiap piksel kanvas terkadang dikenal sebagai **rendering**.

Menggambar Persegi Panjang: Thelurus()Fungsi

Pemrosesan memberi kita kemampuan untuk menggambar persegi panjang di kanvas menggunakan **lurus()** fungsi. Sebagai contoh, mari kita ubah pernyataan kedua dalam program kita menjadi berikut:

ukuran (150, 200);
benar(0, 0, 75, 100);

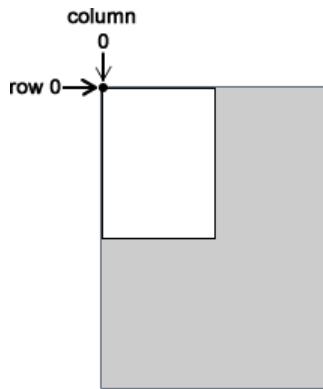
Sekarang, ketika kita menjalankan program kita, kita melihat yang berikut:



Mengapa Processing merender persegi panjang khusus ini? Itu karena *PertamaDanKedua* argumen yang kami berikan kepada *lurus()* fungsi

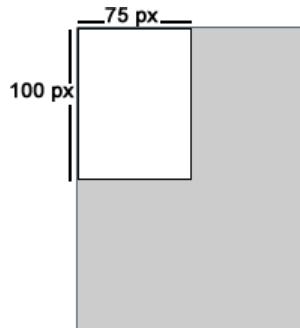
benar(0, 0, 75, 100);

tentukan *kolom* dan *baris*, masing-masing, dari lokasi piksel *pojok kiri atas* dari persegi panjang:

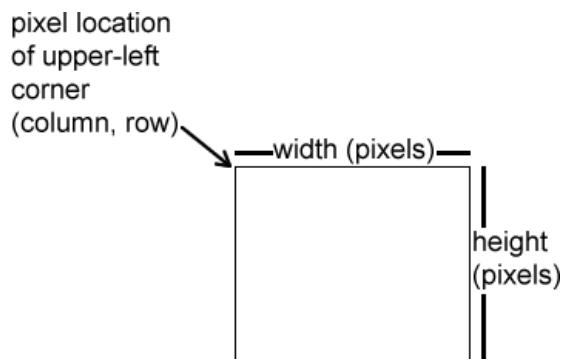


Selain itu, *ketigaDan keempat* argumen yang kami berikan kepada `lurus()` fungsi `lurus(0, 0,75, 100):`

tentukan piksel persegi panjang/*ebardan* piksel *tinggi*, masing-masing:



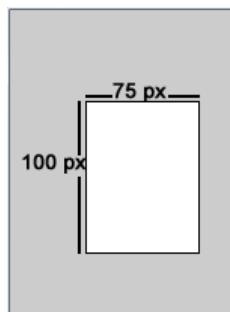
Jadi, secara umum, kita mendeskripsikan persegi panjang dengan `lurus()` berfungsi dengan menentukan lokasi piksel di sudut kiri atas, lebarnya, dan tingginya:



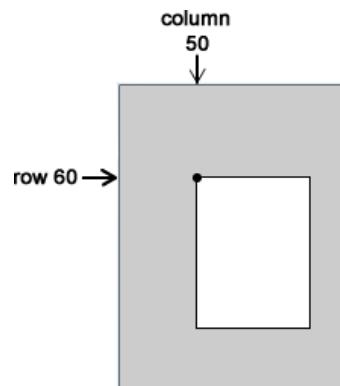
Misalnya, jika kita mengubah *PertamaDan Kedua* argumen dalam panggilan kami kelurus() fungsi

ukuran (150, 200);
benar(**50**,**60**, 75, 100);

kemudian, ketika kita menjalankan program kita, kita melihat sebuah persegi panjang yang ukurannya sama seperti sebelumnya



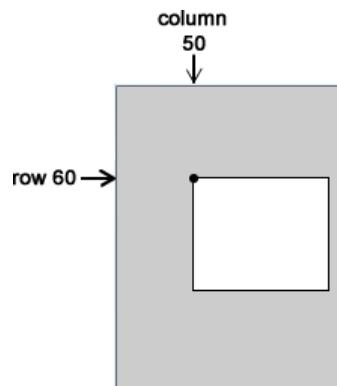
tapi sudut kiri atasnya sekarang berada di kolom piksel 50 dan baris piksel 60:



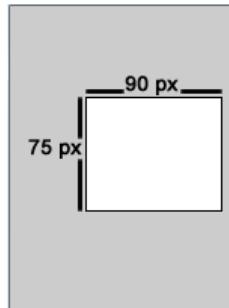
Begitu pula jika kita mengubah *ketiga* dan *keempat* argumen dalam panggilan kami kelurus() berfungsi sebagai berikut,

ukuran (150, 200);
lurus(50, 60,**90, 75**);

kemudian, saat kita menjalankan program, kita masih melihat persegi panjang yang sudut kiri atasnya berada pada kolom piksel 50 dan baris piksel 60:



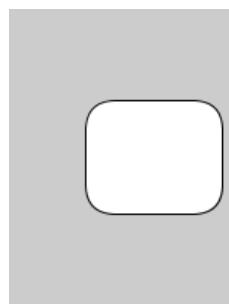
Namun, persegi panjang itu *ebar* sekarang 90 piksel, dan itu *tinggi* sekarang 75 piksel:



Pemrosesan lurus() fungsi juga memiliki opsional *kelima* parameter untuk menggambar *abulat* persegi panjang. Misalnya, jika kita memberikan argumen kelima untuk jumlah pembulatan yang ingin kita terapkan pada setiap sudut persegi panjang,

ukuran (150, 200);
lurus(50, 60, 90, 75, **20**);

lalu, ketika kita menjalankan program, kita malah melihat persegi panjang yang membulat:



Singkatnya, bentuk dasar panggilan *kelurus()* fungsinya adalah

benar(kolom, baris, lebar, tinggi[, pembulatan]);

Parameternya adalah

kolom	Letak kolom piksel di pojok kiri atas persegi panjang
baris	Letak baris piksel di pojok kiri atas persegi panjang
lebar	Lebar persegi panjang dalam piksel
tinggi	Ketinggian persegi panjang dalam piksel
pembulatan	Jumlah pembulatan sudut (opsional)

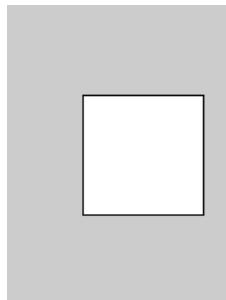
Jadi, dengan informasi sederhana dari kami ini, Pemrosesan mampu merender sebuah persegi panjang.

Perhatikan bahwa, secara default, tepi persegi panjang adalah *hitam*. Warna yang digunakan untuk menggambar titik, garis, dan tepi suatu bentuk disebut **warna guratan**. Perhatikan juga bahwa, secara default, persegi panjang diisi dengan *putih* piksel. Warna yang digunakan untuk mengisi bentuk disebut

itu **mengisi warna**. Kita akan mempelajarinya nanti di bab ini bagaimana mengubah warna guratan dan warna isian.

Tidak ada fungsi dalam Pemrosesan untuk menggambar *apersegi*. Namun, kita dapat dengan mudah menggambar persegi menggunakan *lurus()* fungsi jika kita menentukan lebar dan tinggi persegi panjang dengan jumlah piksel yang sama:

ukuran (150, 200);
*lurus(50, 60, **80, 80**);*



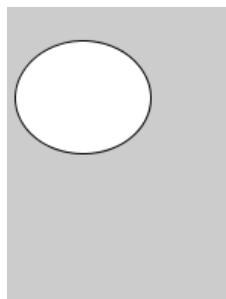
Pilih **File > Simpan** untuk menyimpan kembali program ini. Kami akan terus mengerjakannya di bagian berikutnya.

Menggambar Elips: The *elips()* Fungsi

Pengolahannya juga memudahkan kita menggambar elips menggunakan *elips()* fungsi. Sebagai contoh, mari kita ubah pernyataan kedua dalam program kita menjadi berikut:

ukuran (150, 200);
*elips(50, 60, **90, 75**);*

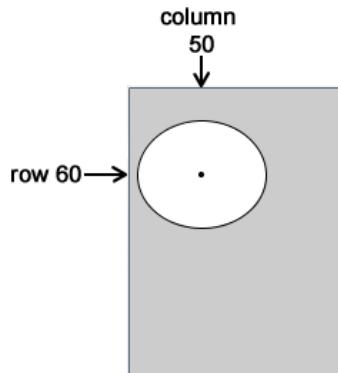
Sekarang, ketika kita menjalankan program kita, Processing mengambil yang berikut ini:



Kami melihat elips khusus ini karena *Pertama Dan Kedua* argumen yang kami berikan kepada *elips()* fungsi

*elips(**50,60**, 90, 75);*

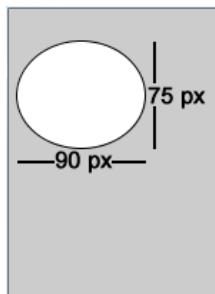
tentukan *kolom* dan *baris*, masing-masing, dari lokasi piksel *tengah* dari elips:



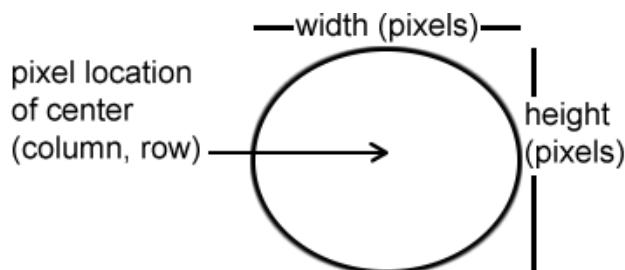
Selain itu, *ketiga* dan *keempat* argumen yang kami berikan kepada *elips()* fungsi

`elips(50, 60, 90, 75);`

tentukan piksel elips/*ebar* dan piksel *tinggi*, masing-masing:



Singkatnya, kami menggambarkan elips dengan *elips()* berfungsi dengan menentukan lokasi piksel pusatnya, lebarnya, dan tingginya:



Jadi, bentuk dasar panggilan ke *elips()* fungsinya adalah

`elips(kolom, baris, lebar, tinggi);`

Parameternya adalah

kolom

baris

Letak kolom piksel pada pusat elips Lokasi

baris piksel pada pusat elips

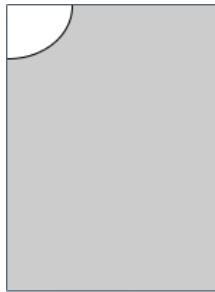
lebar	Lebar elips dalam piksel
tinggi	Tinggi elips dalam piksel

Dengan empat item informasi sederhana dari kami, Pemrosesan mampu membuat elips.

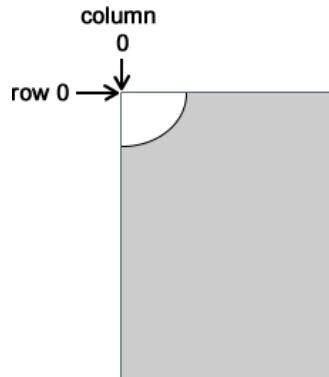
Sekarang, mari kita coba merevisi pernyataan kedua dari program kita sehingga kita menempatkan lokasi pusat elips pada kolom piksel 0 dan baris piksel 0:

ukuran (150, 200);
elips(**0,0**, 90, 75);

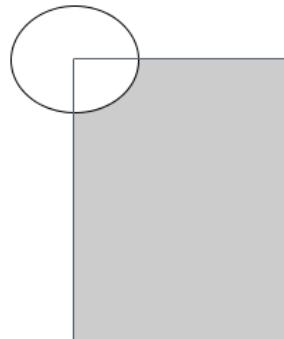
Saat kami menjalankan program kami, sekarang kami melihat yang berikut:



Kita melihat hasil ini karena pusat elips berukuran 90 piksel kali 75 piksel kini terletak di kolom piksel 0 dan baris piksel 0, piksel di sudut kiri atas kanvas:



Jadi, sebagian besar elips yang telah kami tentukan pada dasarnya “tumpah” dari tepi kanvas.

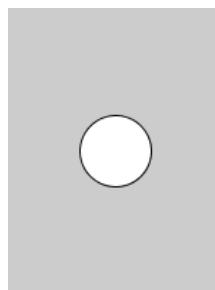


20■Pemrosesan: Pengantar Pemrograman

Perhatikan bahwa Processing tidak memberi kita pesan kesalahan ketika kita memerintahkannya untuk menggambar titik, garis, atau bentuk yang melampaui tepi kanvas. Hal ini memberikan tingkat kenyamanan dan fleksibilitas yang tinggi saat kita berekspeten dengan menggambar. Kami bahkan diizinkan untuk menggunakan *negatif* angka ketika kita ingin menentukan kolom piksel dan/atau baris piksel yang berada di luar kanvas yang terlihat.

Perhatikan bahwa tidak ada fungsi dalam Memproses untuk menggambar *alingkaran*. Namun, kita dapat dengan mudah menggambar lingkaran menggunakan *elips()* berfungsi hanya dengan menentukan lebar dan tinggi elips dengan jumlah piksel yang sama. Misalnya, program berikut menggambar lingkaran dengan diameter 50 piksel di tengah kanvas:

```
ukuran (150, 200);  
elips(75, 100, 50, 50);
```



Pilih **File > Simpan** untuk menyimpan program ini. Kami akan terus mengerjakannya di bagian berikutnya.

Menggambar Segitiga: The *segitiga()* Fungsi

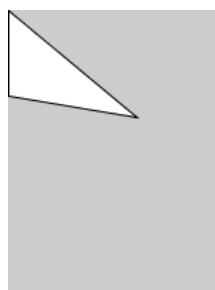
Mari kita pelajari tentang fungsi bentuk dasar lainnya yang disediakan oleh Processing: *the segitiga()* fungsi.

Itu *segitiga()* fungsi memungkinkan kita menggambar segitiga hanya dengan menentukan *tiga* lokasi piksel pada kanvas, masing-masing sesuai dengan salah satu dari tiga simpul (titik sudut) segitiga itu.

Misalnya, mari kita ubah baris kedua kita Menggambar program sebagai berikut:

```
ukuran (150, 200);  
segitiga(0,0, 0, 60, 90, 75);
```

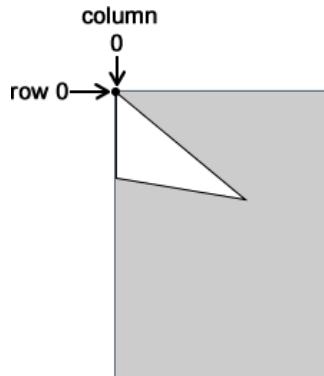
Saat kami menjalankan program kami, kami melihat yang berikut:



Pemrosesan menggambar segitiga khusus ini karena *Pertama Dan Kedua* argumen yang kami berikan kepada *segitiga()* fungsi

segi tiga(**0,0**, 0, 60, 90, 75);

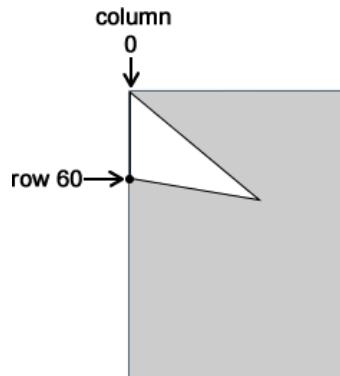
tentukan *kolom* dan *baris*, masing-masing, dari *Pertama* titik sudut (titik sudut) segitiga:



Begitu pula dengan *ketiga* dan *keempat* argumen yang kami berikan kepada *segitiga()* fungsi

segitiga(0, 0,**0,60**, 90, 75);

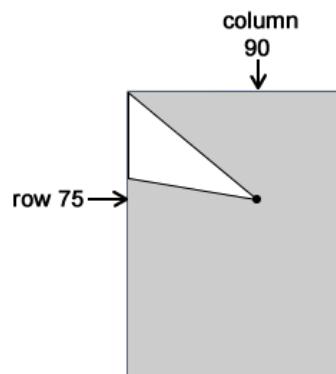
tentukan *kolom* dan *baris*, masing-masing, dari *Kedua* titik sudut segitiga:



Dan yang terakhir adalah *kelima* dan *keenam* argumen yang kami berikan kepada *segitiga()* fungsi

segitiga(0, 0, 0, 60,**90,75**);

tentukan *kolom* dan *baris*, masing-masing, dari *ketiga* titik sudut segitiga:



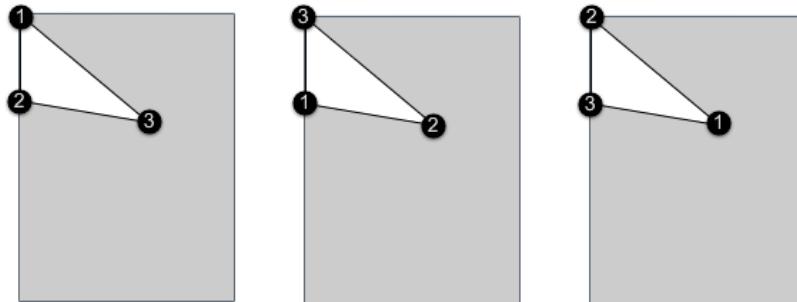
Singkatnya, bentuk dasar dari **segi tiga()** fungsinya adalah

segi tiga(kolom₁, baris₁, kolom₂, baris₂, kolom₃, baris₃);

Parameternya adalah

kolom₁	Lokasi kolom piksel pada simpul pertama
baris₁	Lokasi baris piksel pada simpul pertama
kolom₂	Lokasi kolom piksel pada simpul kedua Lokasi
baris₂	baris piksel pada simpul kedua Lokasi kolom
kolom₃	piksel pada simpul ketiga Lokasi baris piksel
baris₃	pada simpul ketiga

Perhatikan bahwasanaya yang sama akan ditarik terlepas dari itu mempersiapkan mana lokasi piksel dari tiga simpul dicantumkan. Dengan kata lain, tidak masalah lokasi piksel mana yang dianggap sebagai titik pertama, kedua, atau ketiga:



Pilih **File > Simpan** untuk menyimpan program ini. Kami akan terus mengerjakannya di bagian berikutnya.

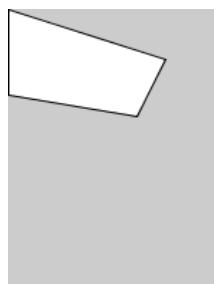
Menggambar Segi Empat : The **segi empat()** Fungsi

Pemrosesan **segi empat()** fungsi memungkinkan kita menggambar segiempat hanya dengan menentukan *empat* lokasi piksel di kanvas.

Misalnya, jika kita mengubah pernyataan kedua dari program kita saat ini menjadi

ukuran (150, 200);
segi empat(0, 0, 0, 60, 90, 75, 110, 35);

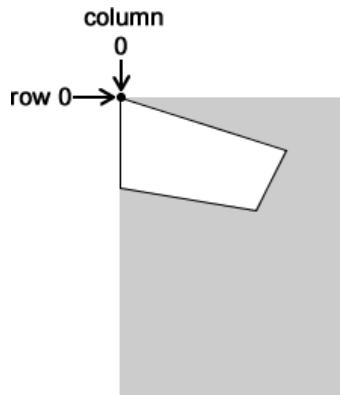
kemudian, ketika kita menjalankan program kita, kita melihatnya



Pemrosesan menggambar segiempat khusus ini karena *Pertama* dan *Kedua* argumen yang kami berikan kepada *segi empat()* fungsi

segi empat(0,0, 0, 60, 90, 75, 110, 35);

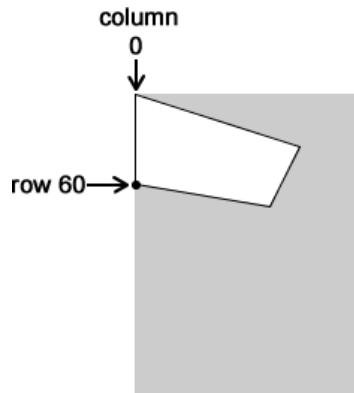
tentukan *kolom* dan *baris*, masing-masing, dari lokasi piksel untuk *Pertama* titik sudut segi empat:



Itu *ketiga* dan *keempat* argumen yang kami berikan kepada *segi empat()* fungsi

segi empat(0, 0, 0, 60, 90, 75, 110, 35);

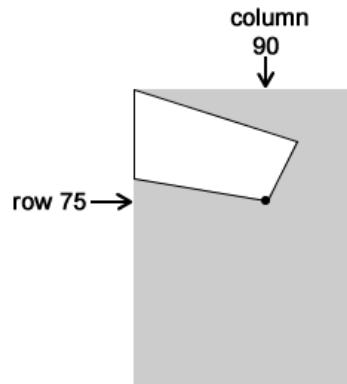
tentukan *kolom* dan *baris*, masing-masing, dari lokasi piksel untuk *Kedua* titik sudut segi empat:



Itu *kelima* dan *keenam* argumen yang kami berikan kepada *segi empat()* fungsi

segi empat(0, 0, 0, 60, 90, 75, 110, 35);

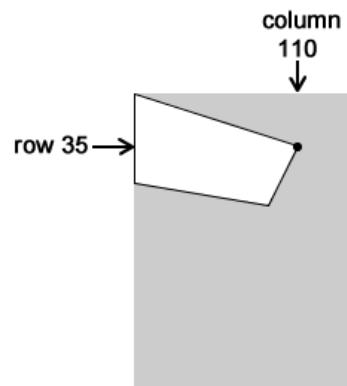
tentukan *kolom* dan *baris*, masing-masing, dari lokasi piksel untuk *ketiga* titik sudut segi empat:



ItuketujuhDankedelapanargumen yang kami berikan kepadasegi empat()fungsi

segi empat(0, 0, 0, 60, 90, 75,110,35);****

tentukankolomDanbaris, masing-masing, dari lokasi piksel untukkeempattitik sudut segi empat:



Singkatnya, bentuk dasar darisegi empat()fungsinya adalah

segi empat(*kolom*₁,*baris*₁,*kolom*₂,*baris*₂,*kolom*₃,*baris*₃,*kolom*₄,*baris*₄);

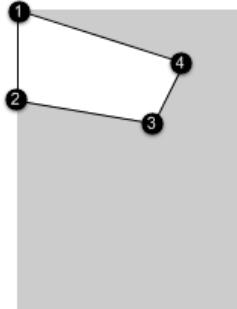
Parameternya adalah

<i>kolom</i>₁	Lokasi kolom piksel pada simpul pertama
<i>baris</i>₁	Lokasi baris piksel pada simpul pertama
<i>kolom</i>₂	Lokasi kolom piksel pada simpul kedua Lokasi baris piksel pada simpul kedua Lokasi kolom
<i>baris</i>₂	piksel pada simpul ketiga Lokasi baris piksel pada simpul ketiga
<i>kolom</i>₄	Letak kolom piksel pada simpul keempat
<i>baris</i>₄	Lokasi baris piksel pada simpul keempat

Dalam kasus tersebut segi empat() fungsi, itu *memesandi* mana simpul terdaftar *menganak* membuat perbedaan. Misalnya, mengingat urutan simpul yang saat ini dicantumkan dalam panggilan kesegi empat() berfungsi dalam pernyataan kedua program kami,

ukuran (150, 200);
 segi empat(0, 0, 0, **60, 90, 75, 110, 35**);

urutan keempat simpul kita pada saat digambar segiempat dianggap



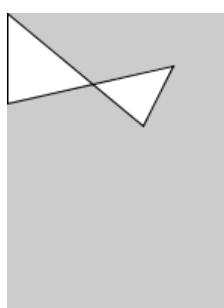
Jika digambar segiempat, maka empat ruas garis yang tergambar menghubungkan keempat titik sudut tersebut adalah

- Dari ① ke ②
- Dari ② ke ③
- Dari ③ ke ④
- Dari ④ ke ①

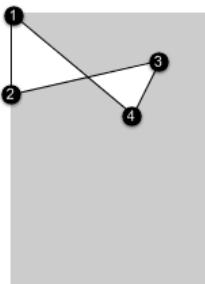
Oleh karena itu, jika kita mengubah urutan simpul dalam pemanggilan fungsi kita,

ukuran (150, 200);
 segi empat(0, 0, 0, **60, 110, 35, 90, 75**);

kemudian, ketika kita menjalankan program yang telah direvisi, kita sekarang melihat bentuk "dasi kupu-kupu":



Ini karena urutan simpul kita berubah menjadi



dan empat ruas garis yang menghubungkan keempat simpul digambar sesuai dengan itu.

Untuk menghindari efek dasi kupu-kupu, cukup mulai daftar argumen kolom dan baris pada simpul tertentu dan lanjutkan dalam urutan searah jarum jam (atau berlawanan arah jarum jam) hingga kolom dan baris dari ketiga simpul lainnya telah terdaftar.

Pilih **File > Simpan** untuk menyimpan program ini. Kami akan terus mengerjakannya di bagian berikutnya.

Menggambar Busur: Thebusur()Fungsi

Pemrosesan **busur()** fungsi memungkinkan kita menggambar *pors/dari* elips.

Ingatlah bahwa bentuk dasar dari **elips()** fungnsinya adalah

elips(kolom,baris,lebar,tinggi);

di mana keempat parameter tersebut berada

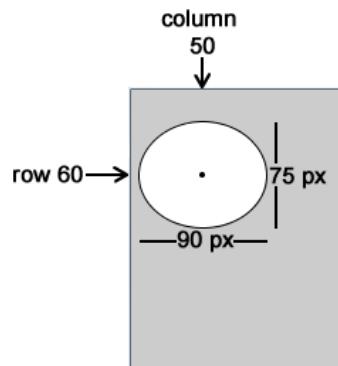
kolom	Letak kolom piksel pada pusat elips Letak
baris	baris piksel pada pusat elips Lebar elips
lebar	dalam piksel
tinggi	Ketinggian elips dalam piksel

Jadi, panggilan ke **elips()** berfungsi pada program berikut,

ukuran (150, 200);

elips(50, 60, 90, 75);

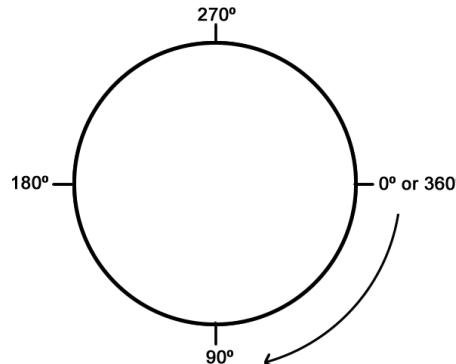
menggambar elips dengan pusatnya terletak di kolom 50 dan baris 60, lebar 90 piksel, dan tinggi 75 piksel:



Empat parameter pertama dari **busur()** fungsinya sama dengan fungsi **elips()** fungsi. Namun, ada dua parameter tambahan, **asudut awal** dan sebuah **sudut akhir**, yang menentukan bagian elips yang akan digambar.

Besaran sudut untuk sudut awal dan akhir harus diberikan *radian* bukannya derajat. Untungnya, Pemrosesan menyediakan **radian()** berfungsi untuk mengubah derajat ke radian.

Penting juga untuk dicatat bahwa pengukuran sudut dilakukan dalam *searah jarum jam* dengan cara yang berlawanan dengan arah jarum jam yang mungkin pernah Anda pelajari di kelas matematika.*

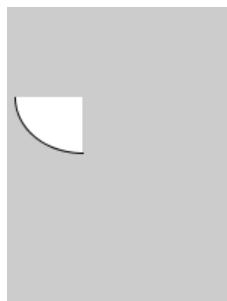


Misalnya, mari kita ubah panggilan kita menjadi **elips()** fungsi ke panggilan berikut ke **busur()** fungsi hanya dengan mengubah nama fungsi dan dengan menambahkan masing-masing sudut awal dan sudut akhir:

ukuran (150, 200);
busur(50, 60, 90, 75, radian(90),radian(180));

Perhatikan bahwa kita dapat menentukan dua sudut menggunakan derajat, mengubah setiap pengukuran sudut menggunakan **radian()** fungsi. Akibatnya, kini ada dua panggilan ker **radian()** fungsi di dalam panggilan kami ke **busur()** fungsi. Kami diizinkan melakukan ini! Memasukkan panggilan ke suatu fungsi di dalam panggilan lain ke fungsi seperti ini dikenal sebagai panggilan fungsi "bersarang".

Saat kami menjalankan program kami, sekarang kami melihat hasil berikut:



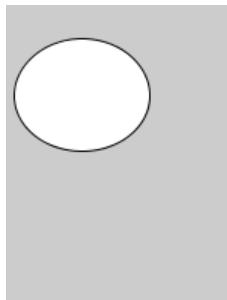
* Ini karena jika kita menganggap kolom dan baris piksel pada kanvas seperti itu *X* dan *kamus* sumbu dalam geometri, lalu *kamu* sumbunya "terbalik" dibandingkan dengan geometri, karena kita bergerak ke bawah pada kanvas, bukan ke atas *kamu* nilai meningkat.

28■Pemrosesan: Pengantar Pemrograman

Kami melihat hasil ini karena empat argumen pertama,

busur(**50,60,90,75**, radian(90), radian(180));

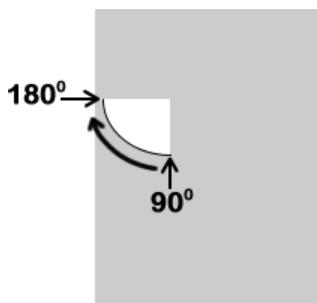
tentukan elips,



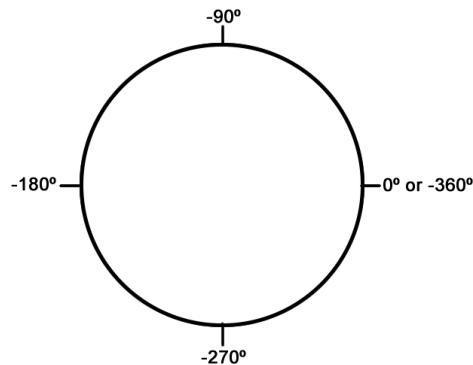
dan dua argumen terakhir,

busur(50, 60, 90, 75, **radian(90),radian(180)**);

tentukan bahwa kita hanya ingin bagian elips ini yang berada di antara posisi 90° dan posisi 180° yang akan digambar:



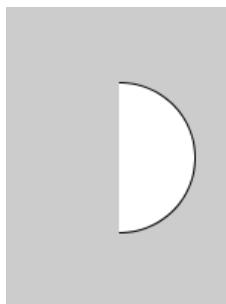
Kita juga dapat mendeskripsikan sudut menggunakan *negatif* derajat



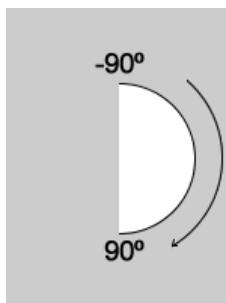
sepanjang sudut awal dan sudut akhir menggambarkan *searah jarum jam* rotasi. Misalnya, panggilan berikut kebusur()fungsi,

```
busur(50, 60, 90, 75,radian(-90),radian(90));
```

akan membuat busur berikut:



Kita melihat hasil ini karena sudut awal ditetapkan sebesar -90° dan sudut akhir ditetapkan sebesar 90° . Penggunaan ukuran sudut negatif diperbolehkan karena kedua sudut tersebut masih ditentukan putarannya searah jarum jam.



Singkatnya, bentuk dasar dari **busur()** fungsi dapat ditulis sebagai

```
busur(kolom,baris,lebar,tinggi,awal,akhir);
```

di mana keenam parameter tersebut berada

kolom	Letak kolom piksel pada pusat elips
baris	Letak baris piksel pada pusat elips
lebar	Lebar elips dalam piksel
tinggi	Ketinggian elips dalam piksel
awal	Sudut permulaan busur (dinyatakan dalam radian)
akhir	Sudut akhir busur (dinyatakan dalam radian)

Pilih **File > Simpan** untuk menyimpan kembali Menggambar program.

Ringkasan

Pada bagian ini, kita mulai dengan mempelajari sejumlah fungsi menggambar elemen grafis di Processing.

ukuran(*lebar,tinggi*);

Parameter

lebar	Lebar piksel kanvas yang diinginkan
tinggi	Tinggi piksel kanvas yang diinginkan

titik(*kolom,baris*);

Parameter

kolom	Kolom piksel tempat titik akan digambar
baris	Baris piksel tempat titik akan digambar

garis(*kolom₁,baris₁,kolom₂,baris₂*);

Parameter

kolom₁	Lokasi kolom piksel pada titik akhir pertama
baris₁	Lokasi baris piksel pada titik akhir pertama
kolom₂	Lokasi kolom piksel pada titik akhir kedua Lokasi
baris₂	baris piksel pada titik akhir kedua

benar(*kolom,baris,lebar,tinggi[,pembulatan]*);

Parameter

kolom	Letak kolom piksel di pojok kiri atas persegi panjang
baris	Letak baris piksel di pojok kiri atas persegi panjang Lebar
lebar	persegi panjang dalam piksel
tinggi	Ketinggian persegi panjang dalam piksel
pembulatan	Jumlah pembulatan sudut (opsional)

elips(*kolom,baris,lebar,tinggi*);

Parameter

kolom	Letak kolom piksel pada pusat elips Letak
baris	baris piksel pada pusat elips Lebar elips
lebar	dalam piksel
tinggi	Ketinggian elips dalam piksel

segi tiga(*kolom₁,baris₁,kolom₂,baris₂,kolom₃,baris₃*);

Parameter

kolom₁	Lokasi kolom piksel pada simpul pertama
baris₁	Lokasi baris piksel pada simpul pertama
kolom₂	Lokasi kolom piksel pada simpul kedua Lokasi
baris₂	baris piksel pada simpul kedua Lokasi kolom
kolom₃	piksel pada simpul ketiga Lokasi baris piksel
baris₃	pada simpul ketiga

segi empat(*kolom₁,baris₁,kolom₂,baris₂,kolom₃,baris₃,kolom₄,baris₄*);

Parameter

kolom₁	Letak kolom piksel pada simpul pertama
baris₁	Lokasi baris piksel pada simpul pertama

kolom₂	Lokasi kolom piksel pada simpul kedua
baris₂	Lokasi baris piksel pada simpul kedua
kolom₃	Lokasi kolom piksel pada simpul ketiga
baris₃	Lokasi baris piksel pada simpul ketiga
kolom₄	Letak kolom piksel pada simpul keempat
baris₄	Lokasi baris piksel pada simpul keempat

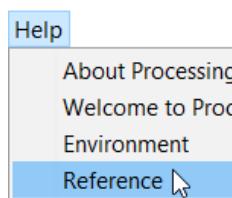
busur(*kolom,baris,lebar,tinggi,awal,akhir*):

Parameter

kolom	Letak kolom piksel pada pusat elips Letak
baris	baris piksel pada pusat elips Lebar elips
lebar	dalam piksel
tinggi	Ketinggian elips dalam piksel
awal	Sudut permulaan busur (dinyatakan dalam radian)
akhir	Sudut akhir busur (dinyatakan dalam radian)

Referensi Pemrosesan

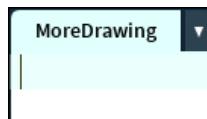
Informasi tambahan tentang semua fungsi di atas dapat ditemukan di**Referensi** yang dibangun dalam Pemrosesan dan dapat dijangkau melalui**Membantu** menu:



Beberapa informasinya agak teknis, jadi jangan khawatir jika Anda tidak memahami semua yang tertulis di Referensi!

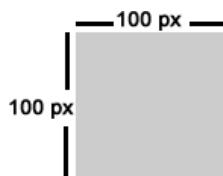
Lebih lanjut tentang Elemen Grafis

Mari pelajari lebih lanjut tentang bekerja dengan elemen grafis di Pemrosesan. Pilih**File > Baru** untuk memulai program baru. Pilih**File > Simpan** dan simpan program baru ini sebagai**Lebih Banyak Gambar**.



(Kamu boleh menutup itu/ainya jendela pemrosesan yang berisi Menggambar program.)

Kita Lebih Banyak Gambar program belum berisi baris kode apa pun. Jadi, ketika kita menjalankannya, kita hanya melihat tampilan yang berisi kanvas abu-abu kosong dengan ukuran default, 100 piksel kali 100 piksel:

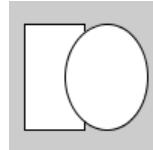


32■Pemrosesan: Pengantar Pemrograman

Di jendela Pemrosesan yang berisiLebih Banyak Gambarprogram, masukkan dua baris kode berikut:

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```

Ketika kita menjalankan program ini, kita melihat bahwa Processing hanya menggambar persegi panjang dan elips pada kanvas dengan ukuran default (100 piksel kali 100 piksel):



Pilih **File > Simpan** untuk menyimpan ini kembaliLebih Banyak Gambarprogram.

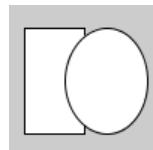
Sepanjang bagian ini, kita akan berulang kali kembali ke dua baris kode ini sebagai titik awal kita.

Urutan Penumpukan

Perhatikan bahwa ketika kita menjalankan program kita saat ini,

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```

elips digambar *diatas dari* persegi panjang.

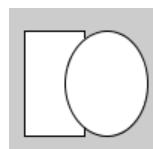


Kami melihat hasil ini karena apa yang terkadang disebut sebagai **urutan penumpukan**. Dalam Pemrosesan, saat elemen grafis digambar *setelah* beberapa elemen grafis lainnya, itu digambar *diatas dari* elemen grafis yang digambar sebelumnya.

Dalam program kami saat ini, panggilan kelurus() fungsi adalah *Pertama* pernyataan dalam program, dan panggilan ke elips() fungsi adalah *Kedua* pernyataan program:

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```

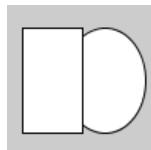
Oleh karena itu, ketika kita menjalankan program ini, elipsnya digambar *setelah* persegi panjang. Oleh karena itu, elips digambar *diatas dari* persegi panjang.



Di sisi lain, misalkan kita *balikurutan* pernyataan dalam program kami:

```
elips(65, 50, 55, 70); persegi(10,
15, 40, 70);
```

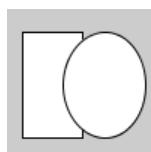
Sekarang, ketika kita menjalankan program kita, persegi panjang digambar setelah elips dan, sebagai hasilnya, persegi panjang digambar di atas dari elips:



Mari kita kembalikan kedua pernyataan kita ke urutan aslinya:

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

Saat kita menjalankan program, elips sekali lagi digambar di atas persegi panjang:



Dalam beberapa perangkat lunak desain grafis, dimungkinkan untuk mengubah urutan penumpukan setelah elemen grafis digambar. Namun, dalam Pemrosesan, satu-satunya cara untuk mengubah urutan penumpukan elemen grafis adalah dengan mengubah urutan gambarnya.

Contoh ini mengilustrasikan prinsip yang sangat penting dalam pemrograman komputer: melakukan tindakan dalam suatu hal tertentu **urutan**. Seperti halnya dalam resep memasak, urutan tindakan individu yang membentuk program komputer dilakukan sangatlah penting. Mengubah urutan tindakan mungkin menghasilkan hasil yang berbeda.

Mengubah Ketebalan Garis: TheBerat pukulan()Fungsi

Secara default, titik, segmen garis, dan tepi di sekitar bentuk yang telah kita gambar semuanya memiliki ketebalan satu piksel. Atribut elemen grafis ini dikenal dalam Pemrosesan sebagai **berat pukulan**.

Pemrosesan menyediakan **Berat pukulan()** fungsi yang memungkinkan kita mengubah bobot guratan yang digunakan ketika titik, segmen garis, atau tepi bentuk selanjutnya digambar.

Bentuk umum dari fungsi ini adalah

berat pukulan(*piksel*);

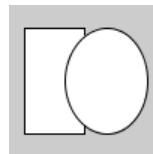
Satu-satunya parameter dari **Berat pukulan()** fungsi adalah

piksel Bobot goresan yang diinginkan, ditentukan dalam piksel

34■Pemrosesan: Pengantar Pemrograman

Pertimbangkan sekali lagi program kami saat ini:

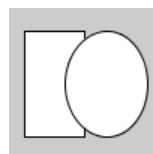
```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```



Jika kita menambahkan pernyataan tersebut

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70); Berat pukulan(10);
```

kemudian ketika kita menjalankan program ini, kita melihat hasil yang sama seperti sebelumnya:

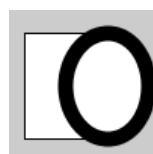


Tidak ada perubahan pada gambar yang dihasilkan Processing karena panggilan kita ke Berat pukulan()saat ini sedang berlangsung *setelah* panggilan kelurus() dan elips() fungsi. Dengan kata lain, kita terlambat mengubah bobot guratan, setelah persegi panjang dan elips digambar. Hanya satu titik, ruas garis, atau tepi bentuk yang digambar *setelah* panggilan ke Berat pukulan()fungsi akan memiliki bobot pukulan baru ini.

Mari kita ubah posisi panggilan keBerat pukulan()berfungsi sedemikian rupa *sebelum* panggilan keelips()berfungsi tetapi *setelah* panggilan kelurus()fungsi:

```
persegi(10, 15, 40, 70);  
Berat pukulan(10);  
elips(65, 50, 55, 70);
```

Sekarang, saat kita menjalankan program, hanya elips yang digambar dengan bobot guratan baru ini:



Sebaliknya jika kita melakukan panggilan keBerat pukulan()berfungsi sebelumnya *keduanya* panggilan kelurus()fungsi dan panggilan keelips()fungsi,

Berat pukulan(10);

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

kemudian, ketika kita menjalankan program kita,*keduanya* persegi panjang dan elips digambar dengan bobot guratan baru ini:



Ada prinsip umum penting yang bekerja di sini: ketika kita mengubah pengaturan untuk gambar elemen grafis, perubahan dalam pengaturan itu akan terjadi.*tetap berlaku* sementara kami menggambar elemen grafis tambahan. Ini sebabnya *keduanya* persegi panjang dan elips digambar dengan bobot guratan baru: bobot guratan saat ini tetap berlaku hingga direset menggunakan panggilan baru ke `Berat pukulan()` fungsi.

Jika kita ingin garis luar elips mempunyai bobot guratan yang berbeda, maka kita cukup menyisipkan pemanggilan kedua ke `Berat pukulan()` fungsi yang datang *setelah* panggilan `kelurus()` berfungsi tetapi *sebelum* panggilan `keelips()` fungsi:

```
Berat pukulan(10);
persegi(10, 15, 40, 70);
```

Berat pukulan(5);

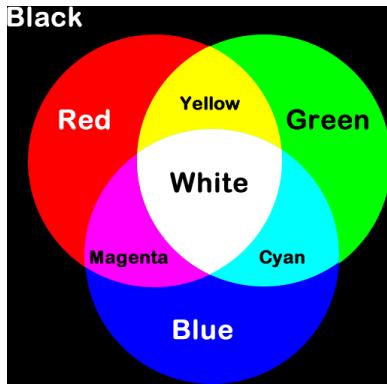
```
elips(65, 50, 55, 70);
```

Sekarang, saat kita menjalankan program, persegi panjang dan elips memiliki bobot guratan yang berbeda:

**Bekerja dengan Warna: RGB**

Sejauh ini, kami hanya menggunakan tiga warna piksel saat menggambar di Pemrosesan: hitam, putih, dan sedikit abu-abu. Namun, Processing memungkinkan kita menggambar dengan warna juga.

Warna harus dijelaskan sebagai *angka* agar komputer dapat bekerja dengannya. Salah satu cara paling umum untuk memberi nomor pada warna adalah berdasarkan **RGB** model warna, di mana semua warna digambarkan sebagai jumlah campuran cahaya merah (R), hijau (G), dan biru (B).



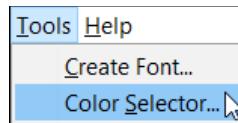
Tiga warna primer cahaya adalah merah, hijau, dan biru. Jika lampu merah dicampur dengan lampu hijau maka akan dihasilkan lampu kuning. Ketika cahaya merah dicampur dengan cahaya biru, dihasilkan cahaya magenta. Ketika cahaya hijau dicampur dengan cahaya biru, cahaya cyan dihasilkan. Ketika cahaya merah, hijau, dan biru dicampur dalam jumlah maksimum, cahaya putih akan dihasilkan. Jika tidak ada cahaya merah, hijau, atau biru sama sekali, maka hasilnya adalah hitam.

Ketika model RGB digunakan dalam teknologi komputer, jumlah cahaya merah, hijau, dan biru biasanya dijelaskan pada skala dari 0 hingga 255. Jadi, warna apa pun dijelaskan dalam RGB sebagai kumpulan tiga angka, yang masing-masing adalah dari 0 hingga 255. Misalnya, angka RGB untuk warna dasar pada diagram di atas adalah sebagai berikut:

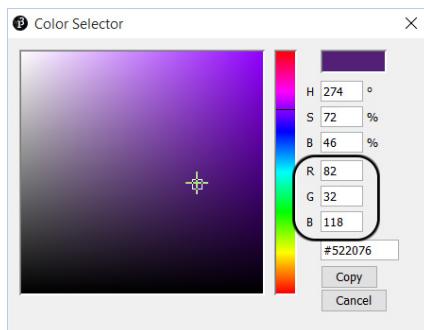
	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Magenta	255	0	255
Cyan	0	255	255
White	255	255	255
Black	0	0	0

Mengapa 255? Jumlah masing-masing komponen merah, hijau, dan biru sebenarnya adalah bilangan biner delapan digit, yang juga dikenal sebagai 1 “byte”. Bilangan biner terbesar yang dapat disimpan dalam satu byte adalah 11111111, yaitu bilangan yang kita tulis dalam desimal sebagai 255.

Pemrosesan juga menyediakan alat bawaan untuk menentukan angka RGB warna tertentu. Dari **Peralatan** menu, pilih **Pemilih Warna**:



Di jendela yang muncul, kita dapat mengklik untuk memilih warna tertentu, dan nilai RGB-nya akan ditampilkan di kotak teks yang sesuai:



Menyetel Ulang Kanvas: The latar belakang()Fungsi

Seperti yang telah kita lihat, warna default kanvas adalah warna abu-abu tertentu. Namun, Processing menyediakan**latar belakang()**fungsii, yang mengatur ulang semua piksel kanvas ke warna RGB apa pun yang kita tentukan.

Bentuk dasar panggilan kelatar belakang()fungsinya adalah

latar belakang(*merah,hijau,biru*);

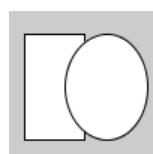
Parameternya adalah

<i>merah</i>	Jumlah warna merah (0 hingga 255)
<i>hijau</i>	Jumlah warna hijau (0 hingga 255)
<i>biru</i>	Jumlah warna biru (0 hingga 255)

Mari kita kembali ke versi awal kitaLebih Banyak Gambarprogram dengan menghapus pernyataan selain yang berikut:

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

Saat kami menjalankan program ini, sekali lagi kami melihat yang berikut:



38■Pemrosesan: Pengantar Pemrograman

Untuk mengubah latar belakang menjadi piksel hijau, kita dapat mencoba yang berikut ini:

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70); latar belakang(0,  
255, 0);
```

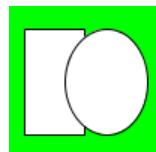
Namun, ketika kita menjalankan program ini, kita melihat kanvas yang seluruhnya terdiri dari piksel hijau:



Mengapa ini hasilnya? Ternyata, itulatar belakang()fungsi agak salah nama. Alih-alih hanya mengubah piksel latar belakang, itulatar belakang()fungsi sebenarnya direset *semua*piksel pada kanvas ke warna yang ditentukan. Untuk itu kita perlu mengatur warna background kanvas *sebelum*kami menggambar di kanvas:

```
latar belakang(0, 255, 0);  
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```

Ketika kita menjalankan program ini, kita sekarang melihat persegi panjang dan elips digambar di atas kanvas hijau:



Mengubah Warna Isi: Themengisi()Dantana Isi()Fungsi

Secara default, Processing mengisi bentuk yang kita gambar—elips, persegi panjang, segitiga, dan segi empat—with piksel putih. Ini dikenal sebagai**mengisi warna**.

Pemrosesan menyediakan**mengisi()**fungsi, yang memungkinkan kita mengubah warna isian bentuk apa pun yang kita gambar setelah kami menyebut fungsi ini.

Bentuk dasar panggilan **kemengisi()**fungsi adalah

```
mengisi(merah, hijau, biru);
```

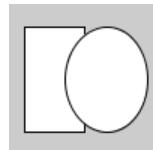
Parameternya adalah

<i>merah</i>	Jumlah warna merah (0 hingga 255)
<i>hijau</i>	Jumlah warna hijau (0 hingga 255)
<i>biru</i>	Jumlah warna biru (0 hingga 255)

Mari kita kembali lagi ke versi awal kitaLebih Banyak Gambarprogram dengan menghapus semua baris kode selain yang berikut ini:

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

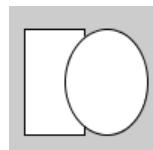
Ingat, saat kita menjalankan program ini, kita melihat yang berikut di kanvas:



Sekarang, misalkan kita ingin elipsnya diisi dengan warna kuning. Untuk mencapai hal ini, kita mungkin mencoba menulis

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70); isi(255, 255, 0);
```

Namun, saat kita menjalankan program, kita melihat hasil yang sama seperti sebelumnya:



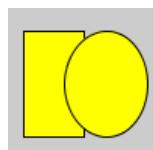
Elips yang digambar tidak diisi dengan warna kuning. Hal ini karena, berdasarkan urutan pernyataan dalam program kita, kita mengatur warna isian menjadi kuning setelah elips sudah digambar. Jadi, apapun tambahan bentuk yang digambar kemudian akan diisi dengan warna kuning, tetapi warna isian bentuk apa pun yang ada sudah telah ditarik akan tetap tidak berubah.

Sekali lagi, kita melihat prinsip umum yang penting bekerja di sini: ketika kita ingin mengubah pengaturan tertentu untuk menggambar elemen grafis, kita harus melakukan ini. *sebelum* kami menggambar elemen grafis yang ingin kami pengaruhi oleh perubahan ini.

Dalam program kita saat ini, kita perlu memindahkan panggilan saat ini kemengisi() berfungsi sedemikian rupa *sebelum* panggilan ke elips() fungsi. Jika kita memposisikan panggilan kemengisi() berfungsi di awal program,

```
isi(255, 255, 0);
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

lalu persegi panjang dan elips diisi dengan warna kuning:

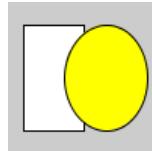


40■Pemrosesan: Pengantar Pemrograman

Jika kita hanya ingin elipsnya diisi dengan warna kuning, maka kita perlu menempatkan panggilan `kemengisi()` fungsi setelah panggilan `kelurus()` berfungsi tetapi sebelum panggilan ke elips() fungsi:

```
persegi(10, 15, 40, 70);  
isi(255, 255, 0);  
elips(65, 50, 55, 70);
```

Sekarang, ketika kita menjalankan program kita, hanya elips yang diisi dengan warna kuning, dan persegi panjang diisi dengan warna isian default, putih:



Penting untuk mengingat prinsip umum yang telah disebutkan sebelumnya:

Saat kita mengubah pengaturan untuk gambar elemen grafis, perubahan pengaturan itu akan terjadi *tetap berlaku* untuk setiap elemen grafis yang kita gambar *kemudian*.

Dalam program saat ini, setiap bentuk yang kita gambar setelah elips akan terus diisi dengan warna kuning, kecuali kita kemudian mengubah pengaturan warna isian menggunakan panggilan `kemengisi()` fungsi atau ketanpa `Isi()` fungsi.

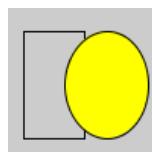
Pemrosesan juga menyediakan **tanpa Isi()** fungsi untuk *matikan* pengisian bentuk. Fungsi ini tidak memiliki parameter apa pun. Jadi, bentuk umum pemanggilan fungsi ini adalah

```
tanpaIsi();
```

Misalnya, mari tambahkan panggilan ketanpa `Isi()` fungsi di awal program kami:

```
tanpaIsi();  
persegi(10, 15, 40, 70);  
isi(255, 255, 0);  
elips(65, 50, 55, 70);
```

Perhatikan bahwa, sebagai hasilnya, persegi panjang sekarang tidak terisi dan, sebagai gantinya, warna default kanvas sekarang terlihat di bagian dalam persegi panjang:



Sebaliknya, perhatikan bahwa elips tetap diisi dengan warna kuning. Sebab, ada panggilan kemengisi()fungsi sebelum kita menggambar elips:

```
tanpaIsi();
persegi(10, 15, 40, 70);

isi(255, 255, 0);
elips(65, 50, 55, 70);
```

Panggilan kemengisi()fungsi tidak hanya mengatur warna isian tetapi juga menyebabkan isian bentuk *melanjutkan*.

Mengubah Warna Goresan: Thestroke()Dantidak ada Pukulan()Fungsi

Perhatikan bahwa semua titik, segmen garis, dan tepi bentuk yang telah kita gambar di Pemrosesan berwarna hitam. Ini karena warna hitam adalah defaultnya**warna guratan**.

Pemrosesan menyediakan**stroke()**fungsi yang memungkinkan kita mengubah warna guratan yang akan digunakan pada setiap titik, ruas garis, atau tepi bentuk yang kita gambar *setelah memanggil fungsi ini*.

Bentuk dasar panggilan ke**stroke()**fungsinya adalah

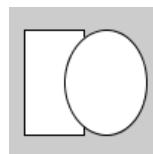
```
stroke(merah,hijau,biru);
```

Parameternya adalah

merah	Jumlah warna merah (0 hingga 255)
hijau	Jumlah warna hijau (0 hingga 255)
biru	Jumlah warna biru (0 hingga 255)

Mari kita revisi sekali lagiLebih Banyak Gambarprogram yang hanya terdiri dari dua baris kode berikut:

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```



Kami sekarang akan menambahkan panggilan keBerat pukulan()fungsi di awal program kami:

```
Berat pukulan(5);
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

42■Pemrosesan: Pengantar Pemrograman

Hasilnya, saat kita menjalankan program ini, tepi di sekitar bentuk kita jauh lebih terlihat:



Selanjutnya, mari kita menelepon `kestroke()`fungsi di awal program kita yang mengatur warna guratan menjadi merah:

```
pukulan(255, 0, 0);
```

```
Berat pukulan(5);
```

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```

Jadi, sebelum kita menggambar bentuk kita, sekarang kita membuat perubahan *dua* pengaturan guratan: guratan **warna** dan pukulannya **berat**. Hasilnya, ketika kita menjalankan program, kita melihat bahwa masing-masing bentuk kita sekarang digambar dengan guratan merah di sekelilingnya yang memiliki ketebalan 5 piksel:



Ingat, perubahan properti gambar tetap berlaku untuk bentuk apa pun yang digambarkan *mudian*. Jadi, dalam program saat ini, setiap goresan—yaitu, menggambar titik, segmen garis, atau tepi bentuk—yang dilakukan *setelah* panggilan ini ke `stroke()` fungsinya juga akan berwarna merah dan ketebalan 5 piksel. Ini sebabnya *keduanya* persegi panjang dan elips telah digambar dengan bobot guratan dan warna guratan yang ditentukan.

Pemrosesan juga menyediakan **tidak ada Pukulan()**fungsi untuk *matikan* gambar tepi di sekitar bentuk. Fungsi ini tidak memiliki parameter apa pun. Jadi, bentuk umum pemanggilan fungsi ini adalah

```
tidak ada Pukulan();
```

Sebagai ilustrasi, mari kita ubah panggilan `menjadistroke()`fungsi di awal program kita untuk panggilan `ketidak ada Pukulan()`fungsi:

```
tidak ada Pukulan();
```

```
Berat pukulan(5);
```

```
persegi(10, 15, 40, 70); elips(65,  
50, 55, 70);
```

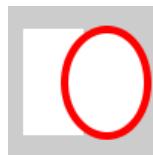
Hasilnya, saat kita menjalankan program, baik persegi panjang maupun elips tidak memiliki tepi di sekelilingnya:



Sekalitidak ada Pukulan()fungsi dipanggil, tidak ada gerakan membela yang akan dilakukan. Tidak akan ada gambar titik, ruas garis, atau tepi bentuk sampai ada panggilan kestroke()fungsi. Misalnya, jika kita membuat perubahan berikut pada program kita,

```
tidak ada Pukulan();
Berat pukulan(5);
persegi(10, 15, 40, 70);
pukulan(255, 0, 0);
elips(65, 50, 55, 70);
```

lalu persegi panjang tersebut masih belum memiliki guratan di sekelilingnya, namun sekali lagi ada guratan merah di sekeliling elips karena digambar setelah panggilan kestroke()fungsi:



Komentar sebaris

Pemrosesan memberi kita pilihan untuk menempatkan komentar dalam kode kita. Salah satu penggunaan umum dari komentar adalah untuk memperjelas tujuan dari satu atau lebih baris kode kita, tidak hanya untuk orang lain yang mungkin membaca kode kita tetapi juga untuk diri kita sendiri saat kita mengerjakan suatu program.

Salah satu bentuk komentar pada Processing with adalah **Di barisan** komentar. Jika kita mengetikkan dua garis miring ke depan (//), Proses akan terjadi mengabaikanapa pun yang kita tulis dari titik di mana garis miring disisipkan hingga akhir baris kode tersebut.

Misalnya untuk memperjelas bahwa panggilan kestroke()fungsi dalam program kami saat ini akan mengaktifkan guratan merah, kami dapat menambahkan komentar berikut:

```
tidak ada Pukulan();
Berat pukulan(5);
persegi(10, 15, 40, 70); pukulan(255, 0, 0);//
guratan merah elips(65, 50, 55, 70);
```

Perhatikan bahwa ini tidak menyebabkan kesalahan, karena Pemrosesan mengabaikan semua yang telah kita tulis mulai dari dua garis miring hingga akhir baris kode tersebut.

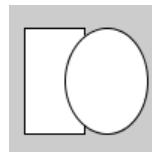
Skala abu-abu

Mari kita edit lagiLebih Banyak Gambarprogram sehingga hanya terdiri dari dua pernyataan berikut:

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

44■Pemrosesan: Pengantar Pemrograman

Saat kita menjalankan program ini, kita sekali lagi melihat gambar berikut di kanvas:

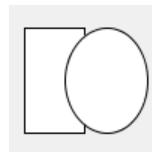


Warna kanvas kami saat ini adalah warna default abu-abu. Ternyata, nilai RGB untuk warna abu-abu ini adalah **204, 204, 204**.

Misalkan kita ingin mengatur latar belakang kita ke warna abu-abu terang. Misalnya, kita dapat melakukannya dengan memasukkan panggilan berikut kelatar belakang()fungsi di awal program kami:

```
latar belakang(240, 240, 240);  
persegi(10, 15, 40, 70); elips(65, 50,  
55, 70);
```

Ketika kita menjalankan program ini, kita sekarang melihat warna abu-abu terang untuk kanvas:



Perhatikan bahwa, sekali lagi, ketiga nilai RGB untuk warna abu-abu ini adalah sama: 240, 240, 240.

Sebenarnya, seperti angka RGB untuk warna hitam—0, 0, 0—and angka RGB untuk warna putih—255, 255, 255—*ketiganya* nomor RGB untuk warna abu-abu selalu *sama*. Setiap kali cahaya merah, hijau, dan biru dicampur dalam jumlah yang sama yaitu lebih besar dari 0 dan kurang dari 255, dihasilkan warna cahaya abu-abu.

Cara lain untuk memvisualisasikan pilihan ini adalah dalam bentuk **skala abu-abu**, mulai dari hitam ke putih melalui banyak corak abu-abu. Dalam Pemrosesan, warna apa pun pada skala abu-abu ini dijelaskan oleh serangkaian tiga angka RGB yang identik, dari 0, 0, 0 (hitam) hingga 255, 255, 255 (putih):



Karena ketiga nomor RGB untuk warna apa pun pada skala abu-abu ini selalu sama, Pemrosesan juga memungkinkan kita merujuk ke hitam, putih, atau bayangan abu-abu tertentu hanya dengan menggunakan *satudari* tiga nomor RGB yang identik. Jadi, dalam contoh kita saat ini, bukan pernyataan kita saat ini

```
latar belakang(240, 240, 240);
```

kami sebenarnya memiliki pilihan untuk sekadar menulis

```
latar belakang(240);
```

Oleh karena itu, kita juga dapat membayangkan angka warna skala abu-abu ini sebagai rentang angka tunggal dari 0 hingga 255.



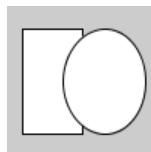
Namun, untuk konsistensi, dalam buku ini kami akan selalu menggunakan sekumpulan tiga angka RGB untuk mendeskripsikan warna apa pun, meskipun warna tersebut hitam, putih, atau abu-abu.

Transparansi

Mari kita kembalikan milik kita sekali lagi! Lebih Banyak Gambar program ke titik awal yang hanya terdiri dari dua baris kode berikut:

```
persegi(10, 15, 40, 70); elips(65,
50, 55, 70);
```

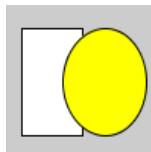
Saat kita menjalankan program ini, kita sekali lagi melihat gambar berikut di kanvas:



Mari kita atur lagi warna isian elips menjadi kuning menggunakan panggilan berikut kemengisi() fungsi:

```
persegi(10, 15, 40, 70);
isi(255, 255, 0);
elips(65, 50, 55, 70);
```

Saat kita menjalankan program ini, warna isian elips disetel menjadi kuning:



Seperti yang telah kita lihat, deskripsi warna RGB terdiri dari tiga angka yang masing-masing angkanya berkisar antara 0 hingga 255. Ini menggambarkan warna yang sepenuhnya **buram**, sama sekali tidak transparan. Jadi, dalam contoh kita saat ini, bagian persegi panjang yang ditumpangi elips sama sekali tidak terlihat.

Namun, Pemrosesan juga memberi kita pilihan untuk menambahkan *a keempat* nomor untuk menggambarkan **transparansi** dari warna yang dipilih. Angka keempat ini juga berkisar antara 0 hingga 255, dimana 0 adalah warna yang benar-benar transparan dan 255 adalah warna yang benar-benar buram (default). Nomor ini terkadang dikenal sebagai warna **kegelapan**, istilah yang mengacu pada bagaimana *buram* warnanya adalah. Ia juga dikenal sebagai *alpha* nilai, dan oleh karena itu, penggunaan empat angka dengan cara ini kadang-kadang dikenal sebagai **RGBA** model warna.

46■Pemrosesan: Pengantar Pemrograman

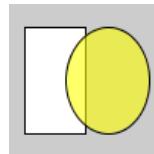
Dalam contoh kita saat ini, panggilan `kemengisi()`fungsi yang mengatur warna isian elips kita

```
isi(255, 255, 0);
```

Jika kita lebih memilih elips kita diisi dengan *asetengah tembus terang*kuning, kita cukup menambahkan *akeempat*argumen untuk panggilan `kemengisi()`fungsi. Argumen keempat ini dapat berupa nilai antara 0 (transparan total) dan 255 (buram total). Misalnya kita bisa menulis

```
persegi(10, 15, 40, 70); isi(255,  
255, 0, 150); elips(65, 50, 55, 70);
```

Sekarang, saat kita menjalankan program, elipsnya diisi dengan warna kuning semitransparan, dan sebagai hasilnya, bagian persegi panjang yang tumpang tindih dengan elips sekarang terlihat sebagian:



Transparansi seperti itu juga dapat digunakan dengan `stroke()`berfungsi untuk membuat titik, ruas garis, atau tepi bentuk yang semitransparan.

Namun saat ini menggunakan model RGBA empat angkat *tidak berpengaruh*dalam Pemrosesan jika digunakan dengan latar belakang()fungsi.

Ringkasan

Di bagian ini, kita mempelajari sejumlah fungsi tambahan yang dapat digunakan untuk menggambar elemen grafis di Pemrosesan.

berat pukulan(*piksel*);

Parameter

piksel Bobot goresan yang diinginkan, ditentukan dalam piksel

latar belakang(*merah,hijau,biru*);

Parameter

merah Jumlah warna merah

hijau Jumlah warna hijau

biru Jumlah warna biru

mengisi(*merah,hijau,biru*);

Parameter

merah Jumlah warna merah

hijau Jumlah warna hijau

biru Jumlah warna biru

tanpaIsi();

Parameter: Tidak ada

stroke(*merah,hijau,biru*);

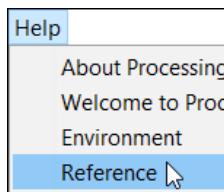
Parameter

<i>merah</i>	Jumlah warna merah
<i>hijau</i>	Jumlah warna hijau
<i>biru</i>	Jumlah warna biru

tidak ada Pukulan();

Parameter: Tidak ada

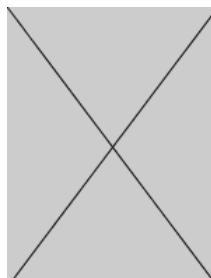
Informasi tambahan tentang semua fungsi ini juga dapat ditemukan di**Referensi** yang dibangun dalam Pemrosesan dan dapat dijangkau melalui**Membantumenu**:



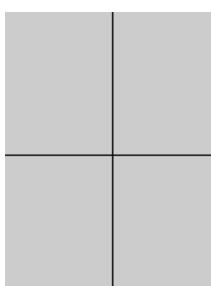
Pemrosesan memiliki lebih banyak kemampuan untuk menggambar daripada yang dibahas dalam bab ini. Namun, Anda sekarang sudah siap untuk memahami contoh grafis di bab-bab selanjutnya. Anda sekarang juga berada dalam posisi yang bagus untuk mulai bersenang-senang bereksperimen dengan menggambar sendiri di Processing.

Latihan

1) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:

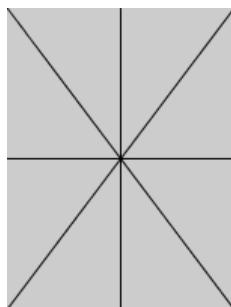


2) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:

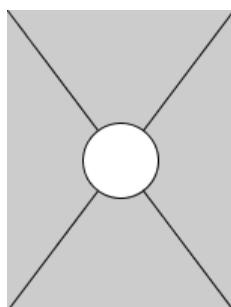


48■Pemrosesan: Pengantar Pemrograman

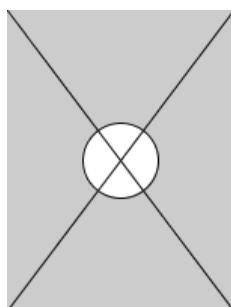
3) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:



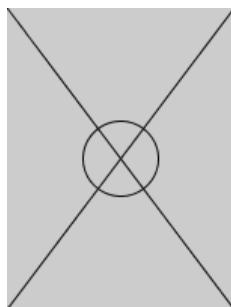
4) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:



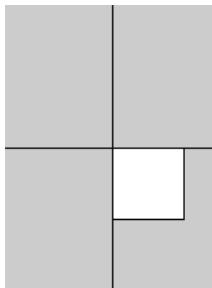
5) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:



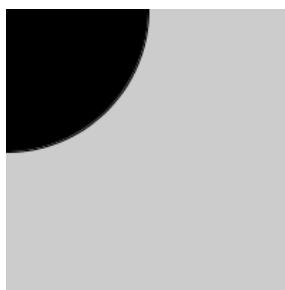
6) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:



7) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 150 piksel kali 200 piksel:



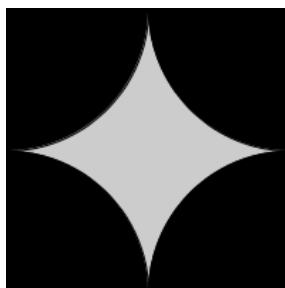
8) Menggunakan elips() atau busur() fungsi, tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



9) Menggunakan elips() atau busur() fungsi, tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:

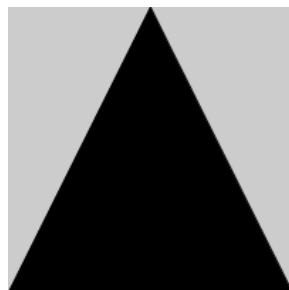


10) Menggunakan elips() atau busur() fungsi, tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:

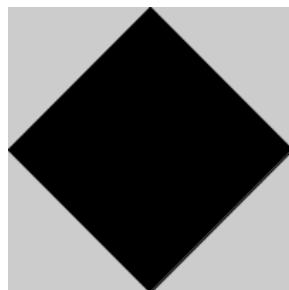


50■Pemrosesan: Pengantar Pemrograman

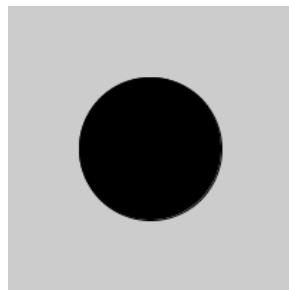
11) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



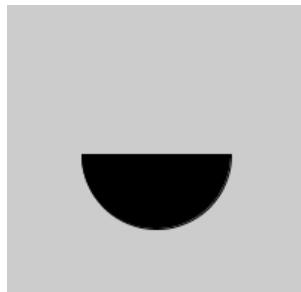
12) Gunakan segi empat()berfungsi untuk menggambar bentuk berlian seperti berikut pada kanvas berukuran 200 pixel x 200 pixel:



13) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



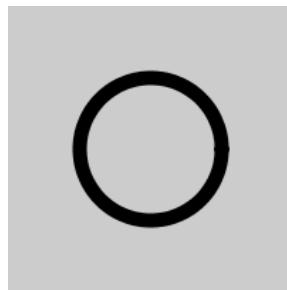
14) Gunakan busur()berfungsi untuk menggambar yang berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



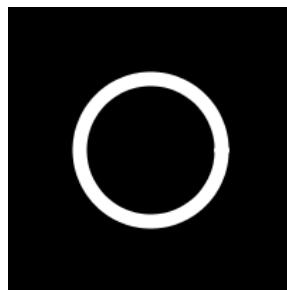
15) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



16) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



17) Tulis kode yang akan menggambar berikut ini pada kanvas berukuran 200 piksel kali 200 piksel:



18) Gambarlah rumah sederhana menggunakan kode Pemrosesan. (Kiat: Gambarlah di kertas grafik terlebih dahulu.)

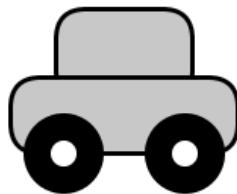
Contoh



52■Pemrosesan: Pengantar Pemrograman

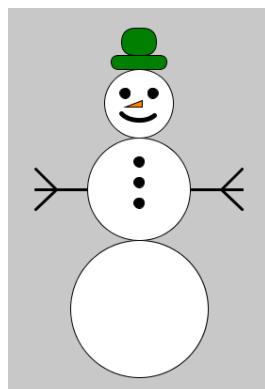
19) Gambarlah mobil mainan menggunakan kode Pemrosesan.

Contoh



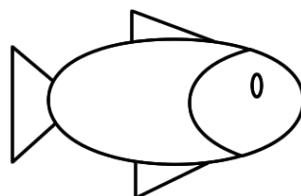
20) Gambarlah manusia salju menggunakan kode Pemrosesan.

Contoh



21) Menggambar binatang menggunakan kode Pemrosesan.

Contoh



22) Buat inisial Anda menggunakan kode Pemrosesan.

23) Pilih bendera nasional dengan desain sederhana dan coba buat ulang dengan kode Pemrosesan.

24) Cobalah untuk membuat ulang bendera Olimpiade dengan kode Pemrosesan.

25) Buat gambar ala pelukis Piet Mondrian menggunakan kode Pemrosesan.