PERTEMUAN 13
STRING DAN MENAMPILKAN TEKS

Kemampuan Akhir yang Diharapkan (KAD):
Mahasiswa mampu memahami grafik string dan text.

Pokok Bahasan:
1. Menyimpan teks dalam objek String
2. Fungsi dasar String
3. Membuat dan memuat font
4. Menampilkan teks
5. Animasi Teks
6. Rotasi Teks
7. Menampilkan teks satu karakter demi satu karakter
8. Soal Latihan

13.1 Dari mana String berasal?

Kelas String bukanlah konsep yang sepenuhnya baru. Anda sudah pernah menggunakan *string* sebelumnya setiap kali Anda mencetak teks ke jendela pesan atau memuat gambar dari sebuah file.

```
println("printing to the message window!");
PImage img = loadImage("filename.jpg");
```

Meskipun demikian, walaupun saya telah menggunakan string di beberapa bagian, saya belum sepenuhnya membahasnya dan menunjukkan potensi lengkapnya. Untuk memahami asal-usul string, mari kita ingat kembali dari mana kelas berasal. Anda tahu bahwa Anda dapat membuat kelas sendiri (seperti *Zoog*, *Car*, dan sebagainya). Anda juga dapat menggunakan kelas bawaan yang sudah tersedia di lingkungan Processing, seperti **PImage**. Dan akhirnya, pada bab sebelumnya Anda telah mempelajari bahwa Anda dapat mengimpor pustaka tambahan Processing untuk menggunakan kelas tertentu seperti **Capture** atau **Movie**.

Meskipun begitu, kelas-kelas tersebut berasal dari kehidupan Anda di dalam “gelembung” Processing. Anda belum menjelajah keluar ke dunia luas yang penuh dengan ribuan kelas Java yang tersedia. Sebelum saya melompat ke API Java, akan sangat berguna untuk sekadar mengintip ke arah itu dan mempelajari salah satu kelas Java yang paling dasar dan fundamental, yaitu kelas **String**, yang akan saya gunakan untuk menyimpan dan memanipulasi teks.

Di mana Anda dapat menemukan dokumentasi untuk kelas String?
Untuk mempelajari detail tentang variabel, fungsi, dan kelas bawaan, referensi Processing selalu menjadi panduan Anda. Meskipun secara teknis merupakan kelas Java, karena kelas String sangat umum digunakan, Processing menyertakan dokumentasinya dalam referensi. Selain itu, tidak diperlukan pernyataan `import`.

Referensi Processing (http://processing.org/reference/String.html) hanya mencakup beberapa metode yang tersedia dari kelas **String**. Dokumentasi lengkapnya dapat ditemukan di situs

Java milik Oracle (http://docs.oracle.com/javase/7/docs/api/java/lang/String.html). Perhatikan juga URL untuk seluruh API Java: (http://docs.oracle.com/javase/7/docs/api/).

13.2 Apa itu String?

Sederhananya, **String** pada dasarnya hanyalah cara yang lebih praktis untuk menyimpan sebuah array karakter. Jika kita tidak memiliki kelas String, kemungkinan besar kita harus menulis kode seperti berikut:

```
char[] sometext = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l',
'd'};
```

Jelas, hal tersebut akan menjadi pekerjaan yang sangat merepotkan di Processing. Akan jauh lebih sederhana jika kita melakukan hal berikut dan membuat sebuah objek **String**:

```
String sometext = "How do I make String? Type some characters
between quotation marks!";
```

Dari contoh di atas, terlihat bahwa sebuah **String** tidak lebih dari sekadar daftar karakter di antara tanda kutip. Namun, itu hanyalah data dari sebuah String. Kita harus ingat bahwa sebuah String adalah sebuah *objek* dengan metode-metode tertentu (yang dapat Anda temukan pada halaman referensi). Ini mirip seperti yang kita pelajari dalam tutorial Pixels, bahwa sebuah **PImage** menyimpan data yang terkait dengan gambar sekaligus menyediakan fungsi-fungsi seperti `copy()`, `loadPixels()`, dan lainnya.
Sebagai contoh, metode **charAt()** mengembalikan karakter tertentu di dalam String pada indeks yang diberikan. Perlu diperhatikan bahwa String bekerja seperti array, di mana karakter pertama berada pada indeks **#0**!

```
String message = "some text here.";
char c = message.charAt(3);
println(c);                // Results in 'e'
```

Metode lain yang berguna adalah **length()**. Metode ini mudah membingungkan dengan properti **length** pada sebuah array. Namun, ketika kita ingin mengetahui panjang sebuah objek String, kita harus menggunakan tanda kurung karena kita sedang memanggil sebuah fungsi bernama **length()**, bukan mengakses sebuah properti bernama **length**.

```
String message = "This String is 34 characters long.";
println(message.length());
```

Kita juga dapat mengubah sebuah String menjadi huruf kapital seluruhnya menggunakan metode **toUpperCase()** (metode **toLowerCase()** juga tersedia).

```
String uppercase = message.toUpperCase();
println(uppercase);
```

Anda mungkin memperhatikan sesuatu yang agak aneh di sini. Mengapa kita tidak cukup menuliskan `message.toUpperCase()` lalu mencetak variabel message? Alih-alih, kita justru

menugaskan hasil `message.toUpperCase()` ke variabel baru dengan nama berbeda—uppercase.
Hal ini karena **String adalah jenis objek khusus**. String bersifat **immutable**. Objek *immutable* adalah objek yang datanya **tidak dapat diubah**. Setelah sebuah String dibuat, ia akan tetap sama selamanya. Ketika kita ingin mengubah String, kita harus menetapkan hasil perubahan tersebut ke variabel (baru atau yang sama).
Jadi, dalam kasus mengubah huruf menjadi kapital, metode `toUpperCase()` **tidak dapat memodifikasi String asli** yang berada dalam message. Metode ini hanya mengembalikan **salinan baru** dari objek String dengan semua huruf kapital.
Perhatikan bahwa kita juga dapat menggunakan variabel message yang sama untuk menyimpan String baru tersebut:

```
message = message.toUpperCase();
```

Anda mungkin berpikir, *"Tunggu, bukankah tadi kamu bilang String itu immutable?"* Ya, benar. Kode di atas tidak mengubah String yang asli. Sebaliknya, kode tersebut memperbarui variabel message agar menunjuk ke objek String baru yang dibuat oleh toUpperCase(). Tanpa penugasan ulang ini (menggunakan operator =), nilai asli dari message akan tetap tidak berubah.
Terakhir, mari kita lihat metode **equals()**. Sekarang, String memang bisa dibandingkan menggunakan operator == seperti berikut:

```
String one = "hello";
String two = "hello";
println(one == two);
```

Namun secara teknis, ketika operator == digunakan untuk objek, operator tersebut membandingkan **alamat memori** dari masing-masing objek. Meskipun dua String memiliki data yang sama—misalnya "hello"—jika mereka merupakan objek yang berbeda, maka perbandingan dengan == dapat menghasilkan nilai **false**.
Fungsi **equals()** memastikan bahwa kita benar-benar memeriksa **apakah dua objek String memiliki urutan karakter yang sama persis**, tanpa peduli di mana data tersebut disimpan di dalam memori komputer.

```
String one = "hello";
String two = "hello";
println(one.equals(two));
```

Meskipun kedua metode di atas menghasilkan hasil yang benar, menggunakan **equals()** lebih aman. Bergantung pada bagaimana objek String dibuat di dalam sebuah sketch, operator == tidak selalu bekerja sebagaimana mestinya.
Fitur lain dari objek String adalah **concatenation**, yaitu menggabungkan dua String menjadi satu. String dapat digabungkan menggunakan operator +. Tentu saja, operator + biasanya berarti menjumlahkan angka. Namun, ketika digunakan pada String, operator tersebut berarti **menggabungkan**.

```
String helloworld = "Hello" + "World";
```

Variabel juga dapat dimasukkan ke dalam sebuah String menggunakan proses penggabungan (concatenation).

```
int x = 10;
String message = "The value of x is: " + x;
```

13.3 Menampilkan Teks

Cara termudah untuk menampilkan sebuah String adalah dengan mencetaknya ke jendela pesan (*message window*). Ini mungkin merupakan sesuatu yang sudah sering Anda lakukan saat debugging. Sebagai contoh, jika Anda ingin mengetahui posisi horizontal mouse, Anda akan menuliskan:

```
println(mouseX);
```

Atau jika Anda perlu memastikan bahwa bagian tertentu dari kode telah dijalankan, Anda mungkin akan mencetak sebuah pesan yang bersifat deskriptif.

```
println("We got here and we're printing out the mouse location!!!");
```

Meskipun ini berguna untuk debugging, hal tersebut tidak akan membantu tujuan kita untuk menampilkan teks kepada pengguna. Untuk menampilkan teks di layar, kita harus mengikuti beberapa langkah sederhana.

1. Deklarasikan sebuah objek bertipe `PFont`.

```
PFont f;
```

2. Buat font dengan mengacu pada nama font dan menggunakan fungsi `createFont()`.

Ini sebaiknya dilakukan hanya sekali, biasanya di dalam `setup()`. Sama seperti ketika memuat gambar, proses memuat font ke dalam memori memakan waktu dan dapat memengaruhi performa program secara signifikan jika ditempatkan di dalam `draw()`.

Anda dapat melihat daftar font sistem yang tersedia menggunakan `PFont.list()`. Karena keterbatasan Java, tidak semua font dapat digunakan dan beberapa mungkin hanya berfungsi di satu sistem operasi tetapi tidak di lainnya.

Ketika berbagi sketch dengan orang lain atau mempublikasikannya di web, Anda mungkin perlu menyertakan versi .ttf atau .otf dari font dalam folder **data** di sketch, karena orang lain mungkin tidak memiliki font yang sama di komputer mereka. Hanya font yang legal untuk didistribusikan yang boleh disertakan.

Selain nama font, Anda juga dapat menentukan ukuran font serta apakah font tersebut ingin menggunakan antialiasing atau tidak.

```
f = createFont("Arial",16,true); // Arial, 16 point, anti-aliasing on
```

3. Tentukan font menggunakan `textFont()`

`textFont()` menerima satu atau dua argumen: variabel font dan ukuran font, di mana ukuran font bersifat opsional. Jika Anda tidak menyertakan ukuran font, maka font akan ditampilkan dengan ukuran yang telah dimuat sebelumnya.

Jika memungkinkan, fungsi `text()` akan menggunakan font *native* daripada font bitmap yang dibuat secara internal oleh `createFont()`, sehingga Anda dapat mengubah ukuran font secara dinamis.

Saat menggunakan renderer **P2D**, sketch akan menggunakan font asli sehingga kualitas gambar dan performa akan meningkat. Namun, dengan renderer **P3D**, yang digunakan adalah versi bitmap dari font, sehingga jika Anda menentukan ukuran font yang berbeda dari ukuran saat font dimuat, teks dapat terlihat pecah atau berpiksel.

```
textFont(f,36);
```

4. Tentukan warna menggunakan `fill()`.

```
fill(255);
```

5. Panggil fungsi `text()` untuk menampilkan teks.

Fungsi ini bekerja seperti menggambar bentuk (*shape*) atau gambar. Fungsi ini membutuhkan tiga argumen—teks yang akan ditampilkan, serta koordinat **x** dan **y** tempat teks tersebut muncul.

```
text("Hello Strings!",10,100);
```

Berikut semua langkah tersebut jika digabungkan:

Contoh 17-1. Simple displaying text

```
PFont f;                              // STEP 1 Declare PFont
variable

void setup() {
  size(200,200);
  f = createFont("Arial",16,true); // STEP 2 Create Font
}

void draw() {
  background(255);
  textFont(f,16);                     // STEP 3 Specify font to be
used
  fill(0);                            // STEP 4 Specify font color
  text("Hello Strings!",10,100);    // STEP 5 Display Text
}
```

Font juga dapat dibuat menggunakan menu **"Tools"** → **"Create Font."** Proses ini akan membuat dan menempatkan file font **VLW** di dalam folder **data** pada sketch Anda, yang kemudian dapat dimuat ke dalam objek **PFont** menggunakan fungsi `loadFont()`.

```
f = loadFont("ArialMT-16.vlw");
```

13.4 Animasi Teks

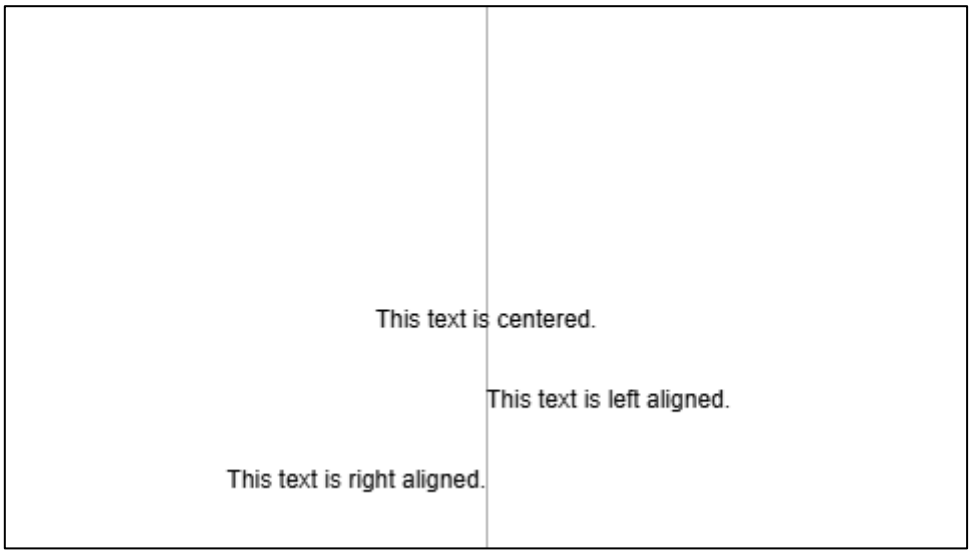
Mari kita lihat dua fungsi Processing lainnya yang berguna untuk menampilkan teks:

`textAlign()` — menentukan perataan teks, apakah **RIGHT**, **LEFT**, atau **CENTER**.

Example 17-2. Text align

```
// Example 17-2. Text align
PFont f;

void setup() {
  size(400,200);
```

```
  f = createFont("Arial",16,true);
}

void draw() {
  background(255);

  stroke(175);
  line(width/2,0,width/2,height);

  textFont(f);
  fill(0);

  textAlign(CENTER);
  text("This text is centered.",width/2,60);

  textAlign(LEFT);
  text("This text is left aligned.",width/2,100);

  textAlign(RIGHT);
  text("This text is right aligned.",width/2,140);
}
```



`textWidth()` — menghitung dan mengembalikan lebar dari sebuah karakter atau string teks. Misalnya, kita ingin membuat *news ticker*, yaitu teks yang berjalan melintasi bagian bawah layar dari kiri ke kanan. Ketika judul berita keluar dari jendela, teks tersebut muncul kembali di sisi kanan dan berjalan lagi. Jika kita mengetahui posisi **x** dari awal teks serta mengetahui lebar teks tersebut, kita dapat menentukan kapan teks tidak lagi terlihat. Fungsi **`textWidth()`** memberikan kita nilai lebar tersebut.

Untuk memulai, kita mendeklarasikan variabel **headline**, **font**, dan lokasi **x**, lalu menginisialisasinya di dalam **`setup()`.**

```
// A headline
String headline = "New study shows computer programming lowers
cholesterol.";
```

```
PFont f;  // Global font variable
float x;  // horizontal location of headline

void setup() {
  f = createFont("Arial",16,true);  // Loading font
  x = width; // initializing headline off-screen to the right
}
```

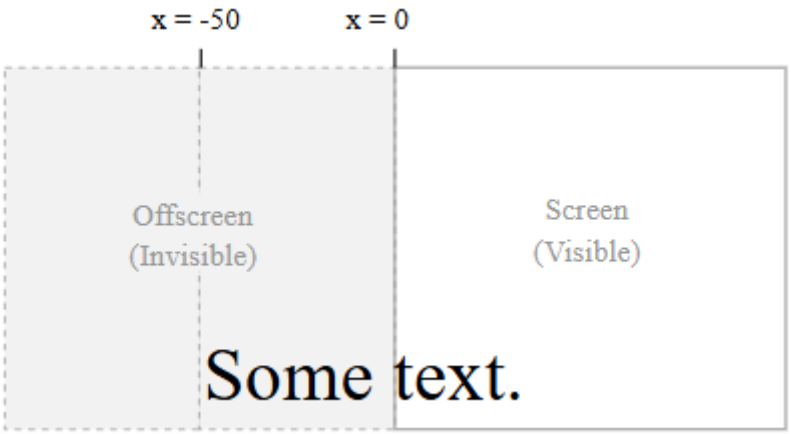
Di dalam `draw()`, kita menampilkan teks pada lokasi yang sesuai.

```
// Display headline at x  location
textFont(f,16);
textAlign(LEFT);
text(headline,x,180);
```

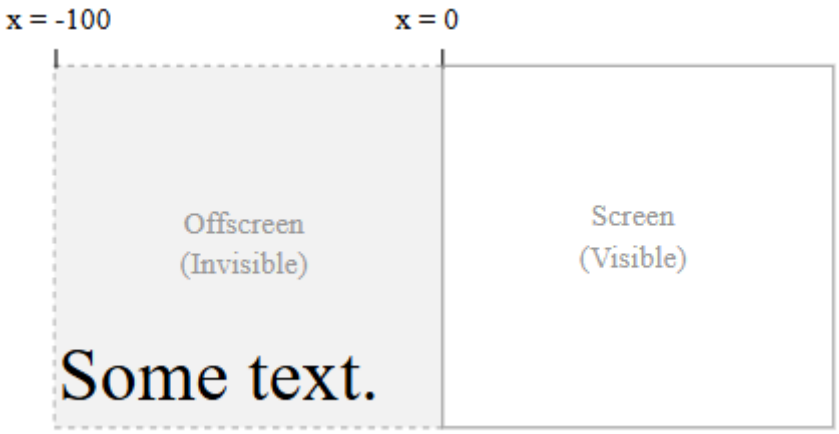
Kita mengubah nilai **x** dengan sebuah nilai kecepatan (*speed*) (dalam contoh ini berupa angka negatif sehingga teks bergerak ke kiri).

```
// Decrement x
x = x - 3;
```

Sekarang kita sampai pada bagian yang lebih sulit. Sebelumnya, mudah untuk mengecek apakah sebuah lingkaran telah mencapai sisi kiri layar — kita cukup menanyakan: *apakah x kurang dari 0?* Namun dengan teks, karena teks rata kiri, ketika nilai **x sama dengan 0**, teks masih tetap terlihat di layar.
Sebaliknya, teks akan menjadi tidak terlihat ketika **x kurang dari 0 dikurangi lebar teks tersebut** (lihat gambar di bawah). Ketika kondisi tersebut terjadi, kita mengatur kembali nilai **x** ke sisi kanan jendela, yaitu **width**.



(a) Sebagian terlihat di layar



(b) Sepenuhnya keluar dari layar

```
// If x is less than the negative width, then it is completely off
the screen
float w = textWidth(headline);
if (x < -w) {
  x = width;
}
```

Berikut adalah contoh lengkap yang menampilkan judul berita (*headline*) yang berbeda setiap kali judul sebelumnya keluar dari layar. Judul-judul tersebut disimpan di dalam sebuah array String.

Example 17-3. Scrolling headlines

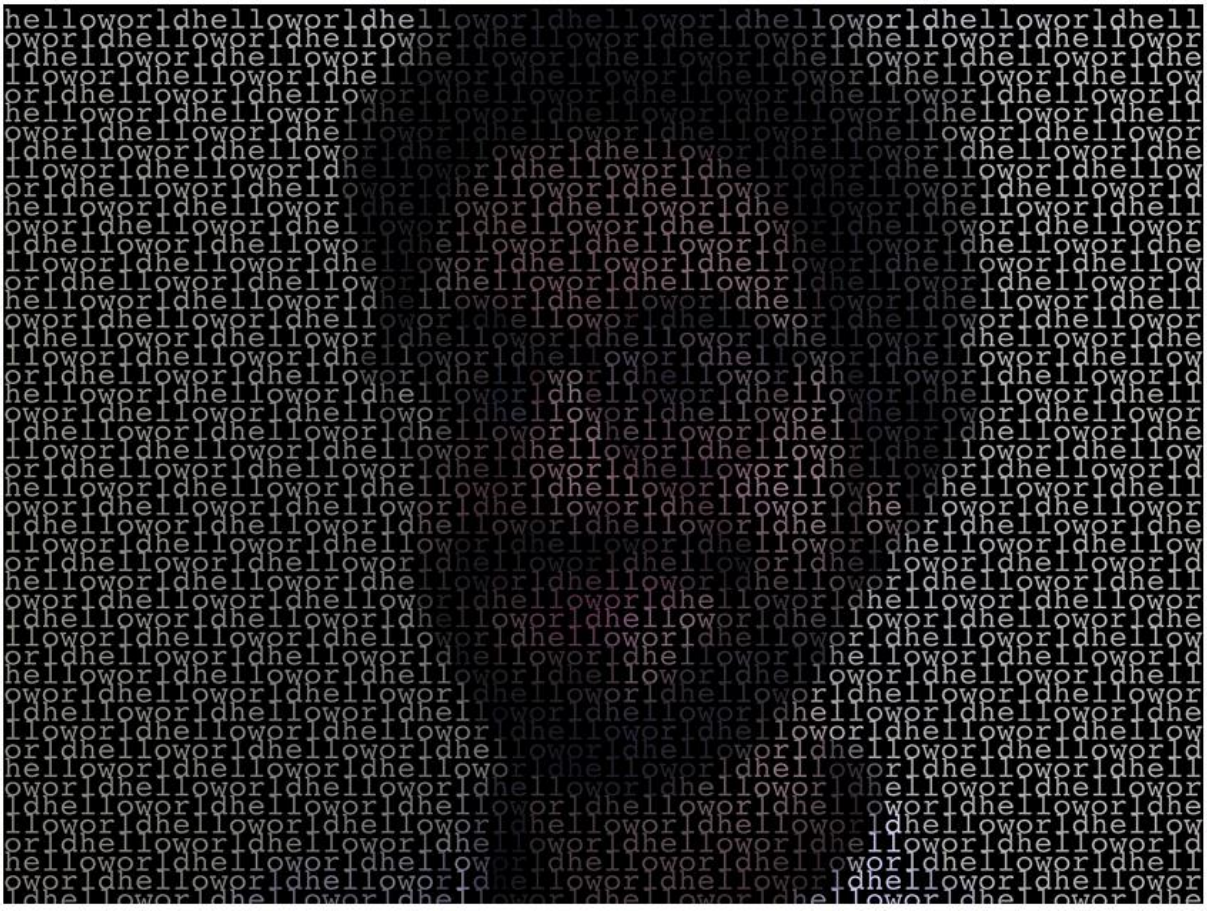
```
// An array of news headlines

String[] headlines = {
  "Processing downloads break downloading record.",
  "New study shows computer programming lowers cholesterol.",
  };

PFont f;  // Global font variable
float x;  // horizontal location of headline
int index = 0;

void setup() {
  size(400,200);
  f = createFont("Arial",16,true);
  // Initialize headline offscreen to the right
  x = width;
}

void draw() {
  background(255);
  fill(0);

  // Display headline at x  location
  textFont(f,16);
  textAlign(LEFT);
  text(headlines[index],x,180);
```

```
  // Decrement x
  x = x - 3;

  // If x is less than the negative width,
  // then it is off the screen
  float w = textWidth(headlines[index]);
  if (x < -w) {
    x = width;
    index = (index + 1) % headlines.length;
  }
}
```

Selain `textAlign()` dan `textWidth()`, Processing juga menyediakan fungsi `textLeading()`, `textMode()`, dan `textSize()` untuk kemampuan tampilan tambahan.

13.5 Mozaik Teks

Dengan menggabungkan apa yang Anda pelajari di **Pertemuan 12** tentang array piksel, Anda dapat menggunakan piksel dari sebuah gambar untuk membuat mosaik yang terdiri dari karakter. Ini adalah pengembangan dari kode *video mirror* pada pertemuan 12. (Perhatikan bahwa pada Contoh 17-4, kode baru yang terkait teks ditampilkan dengan **tebal**.) Lihat Gambar berikut.



Example 17-4. Text mirror

```
// Example 17-4. Text mirror
import processing.video.*;
// Size of each cell in the grid, ratio of window size to video size
int videoScale = 10;
// Number of columns and rows in the system
int cols, rows;
// Variable to hold onto capture object
Capture video;

// A String and Font
String chars = "helloworld";
PFont f;
void setup() {
  size(640, 480);
  // Set up columns and rows
  cols = width / videoScale;
  rows = height / videoScale;
  video = new Capture(this, cols, rows);
  video.start();

  // Load the font
  f = createFont("Courier", 16);
}

void captureEvent(Capture video) {
  video.read();
}

void draw() {
  background(0);
  video.loadPixels();

  // Use a variable to count through chars in a string
  int charcount = 0;

  // Begin loop for rows
  for (int j = 0; j < rows; j++) {
    // Begin loop for columns
    for (int i = 0; i < cols; i++) {
      // Where are you, pixel-wise?
      int x = i * videoScale;
      int y = j * videoScale;

      // Looking up the appropriate color in the pixel array
      color c = video.pixels[i + j* video.width];

      // Displaying an individual character from the String
      // Instead of a rectangle
       textFont(f);
       fill(c);
       text(chars.charAt(charcount), x, y);
       // Go on to the next character
       charcount = (charcount + 1) % chars.length();
```

```
    }
  }
}
```

Teks sumber yang digunakan dalam pola mosaik. String yang lebih panjang mungkin menghasilkan hasil yang lebih menarik.

```
// A String and Font
String chars = "helloworld";
PFont f;
```

Menggunakan font **fixed-width**. Dalam kebanyakan font, setiap karakter memiliki lebar yang berbeda. Pada font fixed-width, semua karakter memiliki lebar yang sama. Hal ini berguna di sini karena kita ingin menampilkan huruf satu per satu dengan jarak yang seragam. Lihat Contoh **17-7** untuk cara menampilkan teks karakter demi karakter menggunakan font yang tidak fixed-width.

```
// Load the font
  f = createFont("Courier", 16);
```

Satu karakter dari teks sumber ditampilkan dengan warna sesuai lokasi piksel. Sebuah variabel penghitung — **"charcount"** — digunakan untuk menelusuri string sumber **satu karakter pada satu waktu**.

```
// Displaying an individual character from the String
      // Instead of a rectangle
       textFont(f);
       fill(c);
       text(chars.charAt(charcount), x, y);
       // Go on to the next character
       charcount = (charcount + 1) % chars.length();
```

13.6 Rotasi Teks

Transformasi seperti translasi dan rotasi juga dapat diterapkan pada teks. Sebagai contoh, untuk memutar teks di sekitar pusatnya, lakukan translasi ke titik asal dan gunakan `textAlign(CENTER)` sebelum menampilkan teks.

Example 17-5. Rotating text

```
// Example 17-5. Rotating text
PFont f;
String message = "this text is spinning";
float theta;

void setup() {
  size(200, 200);
  f = createFont("Arial",20,true);
}

void draw() {
  background(255);
  fill(0);
  textFont(f);                    // Set the font
```

```
  translate(width/2,height/2);  // Translate to the center
  rotate(theta);                // Rotate by theta
  textAlign(CENTER);
  text(message,0,0);
  theta += 0.05;                // Increase rotation
}
```

13.7 Menampilkan teks satu karakter demi satu karakter

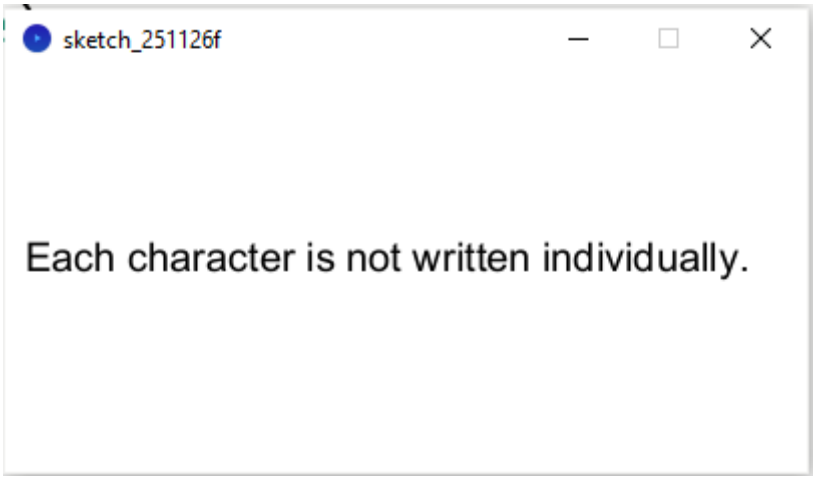
Dalam aplikasi grafis tertentu, menampilkan teks dengan setiap karakter dirender secara terpisah diperlukan. Sebagai contoh, jika setiap karakter perlu bergerak atau diberi warna secara independen, maka hanya menggunakan:

```
text("a bunch of letters",0,0);
```

...tidak akan cukup.

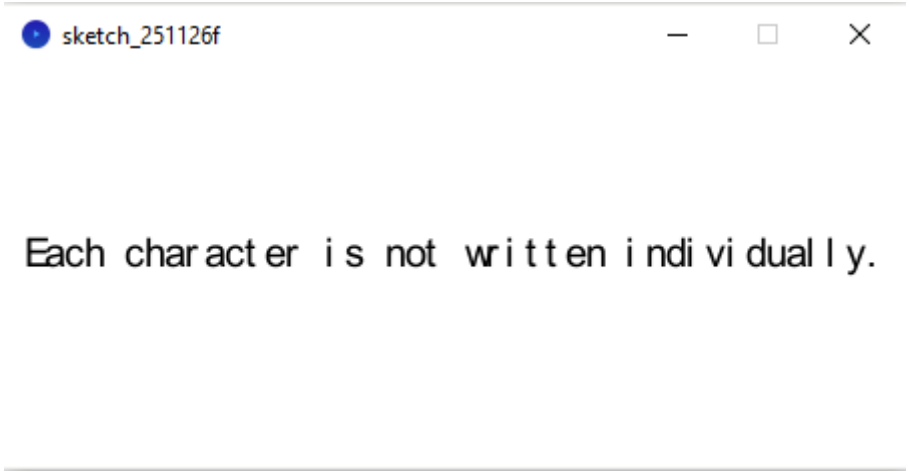
Solusinya adalah melakukan *loop* melalui sebuah String, lalu menampilkan setiap karakter satu per satu. Mari kita mulai dengan melihat contoh yang menampilkan seluruh teks sekaligus.

```
PFont f;
String message = "Each character is not written individually.";

void setup() {
  size(400, 200);
  f = createFont("Arial",20,true);
}

void draw() {
  background(255);
  fill(0);
  textFont(f);
  // Displaying a block of text all at once using text().
  text(message,10,height/2);
}
```



Kita dapat menulis ulang kode untuk menampilkan setiap karakter di dalam sebuah loop dengan menggunakan fungsi `charAt()`.

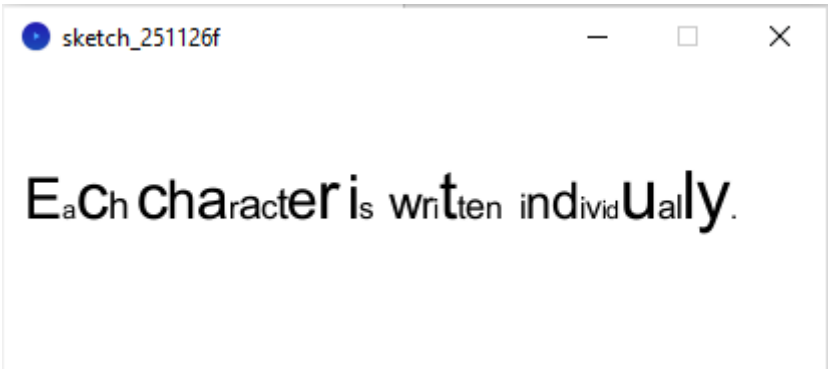
```
String message = "Each character is written individually.";

// The first character is at pixel 10.
int x = 10;
for (int i = 0; i < message.length(); i++) {
  // Each character is displayed one at a time with the charAt()
function.
  text(message.charAt(i),x,height/2);
  // All characters are spaced 10 pixels apart.
  x += 10;
}
```



Memanggil fungsi `text()` untuk setiap karakter akan memberikan lebih banyak fleksibilitas (misalnya untuk pewarnaan, pengaturan ukuran, dan penempatan karakter secara individual dalam satu String). Namun, kode sebelumnya memiliki kelemahan besar — posisi x meningkat sebesar 10 piksel untuk setiap karakter. Meskipun itu kira-kira benar, karena setiap karakter tidak memiliki lebar yang persis sepuluh piksel, jaraknya menjadi tidak akurat.
Jarak yang benar dapat dicapai dengan menggunakan fungsi `textWidth()` seperti yang ditunjukkan pada contoh kode di bawah ini. Perhatikan bahwa contoh ini menghasilkan jarak karakter yang benar bahkan ketika setiap karakter memiliki ukuran acak!

```
PFont f;
String message = "Each character is written individually.";

void setup() {
  size(400, 150);
  f = createFont("Arial",20,true);
}

void draw() {
  background(255);
```

```
  fill(0);
  textFont(f);
  int x = 10;
  for (int i = 0; i < message.length(); i++) {
    textSize(random(12,36));
    text(message.charAt(i),x,height/2);
    // textWidth() spaces the characters out properly.
    x += textWidth(message.charAt(i));
  }
  noLoop();
}
```



Metode “huruf demi huruf” ini juga dapat diterapkan pada sebuah sketch di mana karakter-karakter dari sebuah String bergerak secara independen satu sama lain. Contoh berikut menggunakan desain berorientasi objek (object-oriented design) untuk membuat setiap karakter dari String asli menjadi sebuah objek **Letter**, sehingga memungkinkan karakter tersebut ditampilkan pada posisi yang benar sekaligus bergerak secara individual di dalam layar.

Example 17-6. Text breaking up

```
PFont f;
String message = "click mouse to shake it up";
// An array of Letter objects
Letter[] letters;

void setup() {
  size(260, 200);
  // Load the font
  f = createFont("Arial",20,true);
  textFont(f);

  // Create the array the same size as the String
  letters = new Letter[message.length()];
  // Initialize Letters at the correct x location
  int x = 16;
  for (int i = 0; i < message.length(); i++) {
    letters[i] = new Letter(x,100,message.charAt(i));
    x += textWidth(message.charAt(i));
  }
}

void draw() {
```

```
  background(255);
  for (int i = 0; i < letters.length; i++) {
    // Display all letters
    letters[i].display();

    // If the mouse is pressed the letters shake
    // If not, they return to their original location
    if (mousePressed) {
      letters[i].shake();
    } else {
      letters[i].home();
    }
  }
}

// A class to describe a single Letter
class Letter {
  char letter;
  // The object knows its original "home" location
  float homex,homey;
  // As well as its current location
  float x,y;

  Letter (float x_, float y_, char letter_) {
    homex = x = x_;
    homey = y = y_;
    letter = letter_;
  }

  // Display the letter
  void display() {
    fill(0);
    textAlign(LEFT);
    text(letter,x,y);
  }

  // Move the letter randomly
  void shake() {
    x += random(-2,2);
    y += random(-2,2);
  }

  // Return the letter home
  void home() {
    x = homex;
    y = homey;
  }
}
```

Metode menampilkan karakter satu per satu juga memungkinkan kita untuk menampilkan teks mengikuti bentuk sebuah kurva. Sebelum kita beralih ke huruf, mari kita lihat terlebih dahulu bagaimana cara menggambar serangkaian kotak sepanjang sebuah kurva. Contoh ini sangat bergantung pada penggunaan trigonometri.

Example 17-7. Boxes along a curve

```
PFont f;
// The radius of a circle
float r = 100;
// The width and height of the boxes
float w = 40;
float h = 40;

void setup() {
  size(320, 320);
  smooth();
}

void draw() {
  background(255);

  // Start in the center and draw the circle
  translate(width / 2, height / 2);
  noFill();
  stroke(0);
  // Our curve is a circle with radius r in the center of the
window.
  ellipse(0, 0, r*2, r*2);

  // 10 boxes along the curve
  int totalBoxes = 10;
  // We must keep track of our position along the curve
  float arclength = 0;

  // For every box
  for (int i = 0; i < totalBoxes; i++) {
    // Each box is centered so we move half the width
    arclength += w/2;
    // Angle in radians is the arclength divided by the radius
    float theta = arclength / r;

    pushMatrix();
    // Polar to cartesian coordinate conversion
    translate(r*cos(theta), r*sin(theta));
    // Rotate the box
    rotate(theta);
    // Display the box
    fill(0,100);
    rectMode(CENTER);
    rect(0,0,w,h);
    popMatrix();
    // Move halfway again
    arclength += w/2;
  }
}
```

Yang perlu kita lakukan adalah mengganti setiap kotak dengan satu karakter dari sebuah String yang muat di dalam kotak tersebut. Dan karena karakter tidak semuanya memiliki lebar

yang sama, alih-alih menggunakan variabel **w** yang nilainya tetap, setiap kotak akan memiliki lebar variabel sepanjang kurva sesuai dengan nilai yang diberikan oleh fungsi **`textWidth()`**.

Example 17-8. Characters along a curve

```
// The message to be displayed
String message = "text along a curve";

PFont f;
// The radius of a circle
float r = 100;

void setup() {
  size(320, 320);
  f = createFont("Georgia",40,true);
  textFont(f);
  // The text must be centered!
  textAlign(CENTER);
  smooth();
}

void draw() {
  background(255);

  // Start in the center and draw the circle
  translate(width / 2, height / 2);
  noFill();
  stroke(0);
  ellipse(0, 0, r*2, r*2);

  // We must keep track of our position along the curve
  float arclength = 0;

  // For every box
  for (int i = 0; i < message.length(); i++)
  {
    // Instead of a constant width, we check the width of each
character.
    char currentChar = message.charAt(i);
    float w = textWidth(currentChar);

    // Each box is centered so we move half the width
    arclength += w/2;
    // Angle in radians is the arclength divided by the radius
    // Starting on the left side of the circle by adding PI
    float theta = PI + arclength / r;

    pushMatrix();
    // Polar to cartesian coordinate conversion
    translate(r*cos(theta), r*sin(theta));
    // Rotate the box
    rotate(theta+PI/2); // rotation is offset by 90 degrees
    // Display the character
    fill(0);
    text(currentChar,0,0);
    popMatrix();
```

```
    // Move halfway again
    arclength += w/2;
  }
}
```

13.8 Soal Latihan

1. Apa hasil keluaran dari kode berikut?

```
String message = "a bunch of text here.";
char c = message.charAt(3);
println(c);
```

2. Lakukan loop melalui setiap karakter dari sebuah String dan cetak satu per satu!

```
/*
String message = "a bunch of text here." ;
for (int i = 0; i < _____________ ); i++ ) {
  char c = _________________ ;
  println(c);
}
*/
```

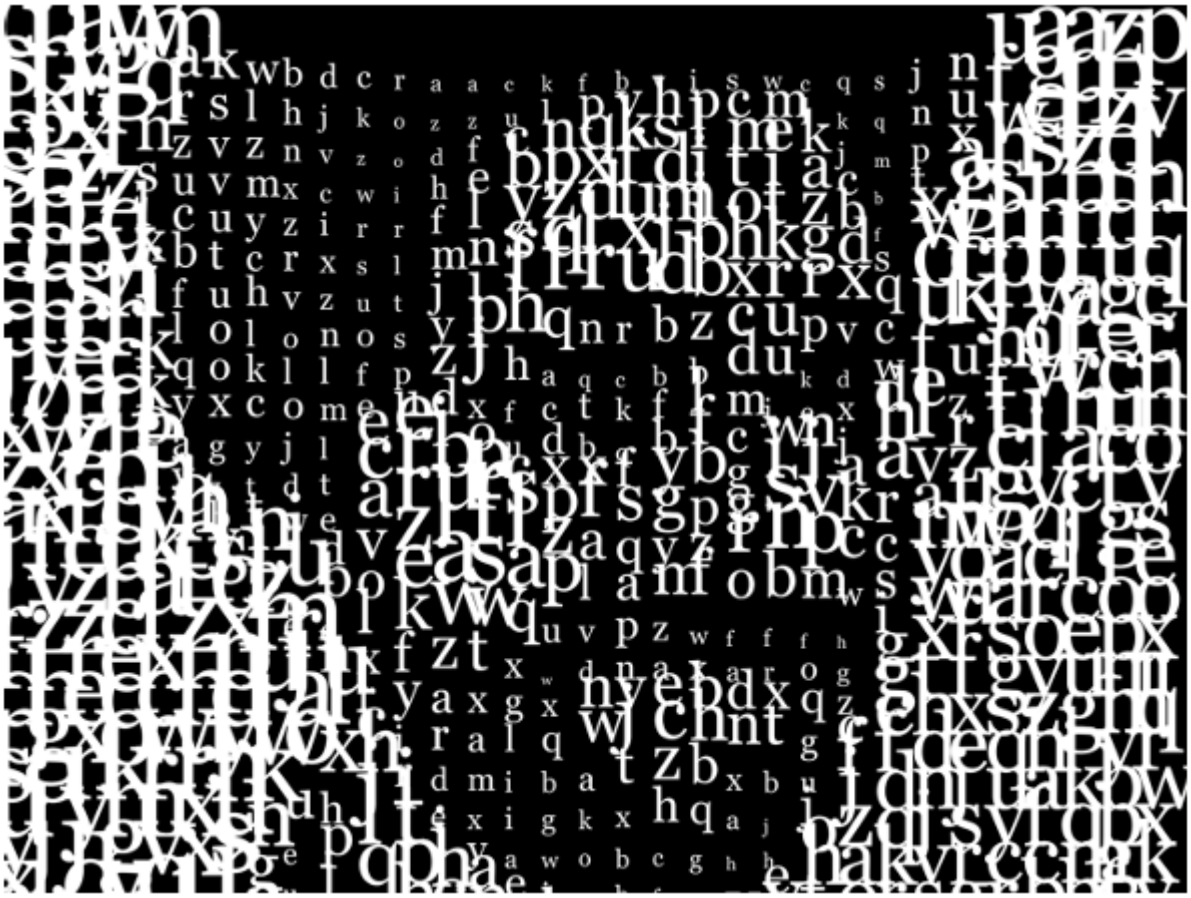
3. Temukan duplikat di dalam array String berikut!

```
/*
String words = { "I" , "love" , "coffee" , "I" , "love" , "tea" } ;
for (int i = 0; i < _____________; i ++ ) {
  for (int j = _; j < _____________; j ++ ) {
    if (_____________________) {
      println(_____________ + " is a duplicate. " );
    }
  }
}
*/
```

4. Gabungkan sebuah string dari variabel yang diberikan sehingga menghasilkan pesan berikut:
   That rectangle is 10 pixels wide, 12 pixels tall and sitting right at (100,100).

```
float w = 10;
float h = 12;
float x = 100;
float y = 100;
String message = ____________________________________;
println(message);
```

5. Buatlah sebuah mosaik teks berbasis video di mana setiap huruf berwarna putih, tetapi ukuran setiap huruf ditentukan oleh tingkat kecerahan piksel. Semakin terang piksel tersebut, semakin besar ukuran hurufnya. Berikut adalah sedikit potongan kode dari bagian dalam loop piksel (dengan beberapa bagian dikosongkan) untuk membantu Anda memulai.



```
float b = brightness(video.pixels[i + j*video.width]);
float fontSize = ____ * (_____ / _____);
textSize(fontSize);
```

Daftar Pustaka:

1. Shiffman, Daniel. 2015. *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction Second Edition*. United States of America: Morgan Kaufmann.
2. https://processing.org/tutorials/text