3 Variablen

Container zum Füllen von Informationen

- Variablen Deklarieren
- Primitive Variablen
- Konvertieren von Variablen
- Gemischte Variablen

3.1 Variablen deklarieren

Processing bietet uns verschieden Variablentypen an, die unterschiedliche Inhalte aufnehmen können. Wie man eine Variable definiert (deklariert) ist ganz einfach. Man überlegt sich den Typ und einen Namen dafür.
Für eine Ganzzahl z.B. `int ganzZahl;`
Jetzt existiert die Variable, wie eine leere Seite eines Buches. D.h. Sie hat noch keinen Wert zugewiesen bekommen. Den Wert einer Variable kann man auch aus anderen Programmabschnitten ändern, sofern sie ihr Gültigkeitsbereich nicht verletzt wird. Denn es gibt lokale und globale Variablen. Den Unterschied werden wir etwas später kennenlernen. Wenn die Variabel einmal deklariert ist muss man ihren Typen (z.B int) bei anderer Wertzuweisung nicht nochmal schreiben, denn dann denkt Processing , dass es hier um eine neue Dekleration handelt. Das sollte dann so aussehen.

```
int ganzZahl;          // Dekleration
....
ganzZahl = 20;         // Zuweisen eines Wertes
....
int ganzZahl = 20;     // ober beides zusammen
```

Um dies in einem kleinen Skript zu verdeutlichen möchte ich gern zwei Ganzzahlen-Variablen definieren, addieren und hinterher ihren Wert in eine Ergebniss-Variable schreiben und auslesen.

Code:

```
// Definieren der Variablen
int ergebnis;              // Ergebniss der Addition
int zahl_1 = 100;          // var 1
int zahl_2 = 100;          // var 2

// Var mit neuen Wert überschreiben
zahl_1 = 2 ;

// Jetzt zusammen addieren und in Ergebnis schreiben
ergebnis = zahl_1 + zahl_2;

// Ergebnis Wert auslesen und im Output darstellen
println("Das Ergebnis: "+ergebnis );
```

Noch etwas zu Namensgebung von Variablen

Bevor du gleich verschiedenste Variablen definierst – Vorsicht!
Variablen Namen müssen:

° Ausschliesslich aus Buchstaben, Nummer oder Unterstrichen generiert werden. (Keine Leerzeichen, Sonderzeichen oder Punkte)

° Mit einem Buchstaben oder Unterstrich anfangen.

° Dürfen nicht länger als 255 Zeichen lang sein. (Okay, wer das diese Länge ausreizt hat irgendwas nicht richtig verstanden:)

° Als case-sensitive betrachtet werden. D.h. für den Kompiler sind die Variablennamen „ganzZahl" oder „ganzzahl" zwei verschieden paar Schuhe!

Legale Variablennamen können so aussehen:

```
int vor_name;
int zaehler;
int sehrLangerVariablenName;
```

Diese Variablennamen würden dich in Probleme rennen lassen:

```
int 1vor_name;                  // Startet mit einer Nummer
int variable mit leerzeichen;  // Enhält Leerzeichen
int auch-illegal;               // Enthält Sonderzeichen
```

Es ist eine gute Übung Variablen gleich am Anfang des Skriptes zu schreiben und zu Kommentieren. So sieht ein Skript dann gleich gut organisiert und schnell verständlich aus:

```
//----------------------------------------
// Initialisierung aller globalen Variablen
//----------------------------------------
float  ballGeschwindigkeit;
int    punktestand;
String spielerName
```

Ok. lassen wir das Gelesene nochmals vor unserem geistigen Auge ablaufen, so fällt uns gleich die generische Struktur auf.

VariablenTyp VariablenName = Wert;
Dies gilt für alle primitiven Variablentypen.

Vorher habe ich zwischen globalen und lokalen Variablen unterschieden. Globale Variablen sind aus jeder stelle des Codes lesbar und setzbar. Sie sollten daher im Kopf des Codes stehen, damit sie sofort gesammelt und übersichtlich zu sehen sind.

Lokale Variablen haben nur eine bestimmte Lebensdauer und werden nach ihrem Benutzen vom Speicher entfernt. Sie stehen auch nur einem begrenzten Programm Abschnitt zur Verfügung. Ein Beispiel wäre hier die for-Schleife:

```
for(int i = 0; i < 10; i++)
{
    ... Anweisung
}
```

Nachdem die Schleife abgearbeitet ist verfällt „i". Es kann dann nicht mehr darauf zugegriffen werden.

Ein anderes Beispiel für lokale Variablen kann man auch anhand von Funktionen sehen. In dem nächsten Beispiel werde ich zwei Variablen gleichen Namens definieren und durch eine Funktion auslesen. Man merkt schnell welchen Gültigkeitsbereich sie haben.
Spiel doch einfach ein bisschen mit dem Code damit es später richtig flutscht :-)

Code:

```
// Definiere global Variable
int zahl = 99;

// Definiere lokale Variable gleichen Namens
// in einer Funktion
void meineFunktion()
{
   int zahl = 11;

   // Zeigt auf die lokale Variable
   println("lokal: "+zahl);
}

void setup()
{
  // Zeigt auf die globale Variable
  println("global: "+zahl);

  // Aufruf der Funtion "meineFunktion"
  meineFunktion();
}
```

3.2 Primitive Variablen

Ich möchte nun hier die verschiedenen primitiven Variablentypen aufführen und ihnen gleich einen Wert zuweisen. Da Processing typisiert ist, dürfen nur die richtigen Werte in eine bestimmte Variabel und nicht einfach so gemischt werden, sonst gibt uns der Kompiler eine Fehlermeldung aus.

° Ganzzahlen `(int)`:
Sie haben einen Gültigkeitsbereich von
2,147,483,647 bis -2,147,483,648.

```
int ganzZahl = 10;
```

° Kommazahlen `(float)`:
Der Datentyp float hat einen Dezimalpunkt - also eine grössere Auflösung als die Ganzzahl. Anderst als bei z.B Java verbrauchen beide Typen 32 bit. Dies liegt daran dass Processing keine andern Typen wie z.B. long, short benutzt, der Einfachheitshalber. Sie haben einen Gültigkeitsbereich von 3.40282347E+38 bis -3.40282347E+38.

```
float dezimalZahl = 10.4;
```

° Boolean (boolean):

Boolsche Werte haben nur zwei Zustände. Nämlich wahr oder falsch. Sie werden oft für Bedingungen oder Programmflüsse gebraucht, um den Code übersichtlicher zu machen.

```
boolean bool = true; // beide möglichen Werte true/false
```

° Characters (char):

Dieser primitive Datentyp steht für typografische Symbole. Damit sind alle Symbole oder Zeichen die der Computer bereithält gemeint. Z.B. das 'm' oder '@'. Jeder char ist 2 Byte lang (16bit) und enthält jeweils ein Zeichen des Unicode-Satzes. Er kann durch ein Zeichen in einzelnen Anführungszeichen beschrieben werden.

```
char myChar = 'a';
```

oder auch durch den alphanummerischen Wert.

```
char myChar = 97; // steht für 'a'
```

° Byte (byte):

In Bytes können alphanummerische Werte von 127 bis -128 gespeichert werden. Sie verbrauchen nur 8bit (1 Byte) an Steicherplatz. Sie können auch zur Repräsentation von chars verwendet werden.

```
byte myByte = 97; // kann für 'a' stehen
```

° Color (color):

In dem Datentyp color werden Farbwerte abgelegt und mit 32bit representiert. Dazu gibt es mehrere Möglichkeiten. Ein Farbwert setzt sich aus R,G,B zusammen (rot, grün, blau). Mit diesem Wert können alle Farben dargestellt werden, die der Bildschirm anzeigen kann. Die Mischung machts! Z.B `color(0,0,0)` steht für schwarz und `color(255,255,255)` steht für weiss. Daraus ergibt sich ein Wertebereich von 0-255. Ich möchte es gern an einem kleinen Beispiel aufzeigen, welches den Hintergrund eines Applets verfärbt.

Code:

```
color bgColor = color(255,0,0); // ergibt rot
background(bgColor);            // setzte BG Farbe
```

Es gibt auch die Möglichkeit den Farbwert in #HEX anzugeben.
Manche kennen das auch aus HTML.

Code:

```
color bgColor = #00FF00; // ergibt grün
background(bgColor);     // setzte BG Farbe
```

Auch den umgekehrten Weg gibt es! Man kann die Farbe eines Pixels auslesen! `get(int xPosition,int yPostition)` Stell dir vor, du möchtest einen Farbwert eines Bildes an einer bestimmten Stelle auslesen, um mit diesem Wert z.B weiter zeichnen zu lassen. Dazu malen wir zwei Kreise an verschiedenen Stellen und färben den Zweiten mit der Farbe des Ersten.

Code:

```
// Zeichne ersten Kreis
color kreisFarbe = #00FF00; // ergibt grün
fill(kreisFarbe);
ellipse(20,width/2,30,30);

// Zeichne zweiten Kreis mit der Farbe von einem
// bestimmten Pixel
color kreisFarbe_2 = get(20,40); // Pixel x:20 y:40
fill(kreisFarbe_2);
ellipse(80,width/2,30,30);
```

3.3 Konvertieren von primitiven Variablen

Es gibt Situationen im Leben eines Programmierers, da möchte man gern z.B. zwei verschieden Variablen miteinander kombinieren und sie dann als ein Ergebnis in einer Dritten speichern. Was passiert aber wenn ich integer (int) mit einem float addieren möchte, um es dann als float zu speichern? Probieren wir es doch aus!

Code:

```
int   a = 10;
float b = 9.9;
float   c;
c = a + b;
println(c); // Ergebniss passt 19.9!
```

Soweit so gut.
Jetzt versuchen wir mal das Gleiche, nur mit dem Unterschied dass die Ergebnisvariable c ein integer.

Code:

```
int   a = 10;
float b = 9.9;
int   c;
c = a + b;
println(c); // FEHLER!!!
```

Dieser Fehler entsteht dadurch, dass das Ergebnis wenn ich a und b addiere ein float ist. Wir wissen dass ein float hochauflösender, da er eine Dezimalstelle besitzt. Da ist es doch ganz klar dass das Resultat nicht in den int c reinpasst.

Auch für solch' spannende Konvertierungen gibt es eine Möglichkeit – namens **Casting**. `(typ)wert` Es ist nichts anderes als ob ich es vom Programm erzwinge ein float in ein int zu stecken. Denn immerhin habe ich die Kontrolle über den Computer und mein Programm! Jedoch fällt dabei leider die Dezimalstelle heraus. Sie hat einfach keinen Platz und wird übersehen.

Code:

```
int   c;
c = 10 + (int)9.9; // ich caste auf (int)9.9 -> 9
println(c);
```

Solche Zahlen sollte man vorher noch zu runden, um bessere Ergebnisse zu bekommen. Aber das besprechen wir im Math Kapitel. Versuche doch zuvor andere „cast" Möglichkeiten von Primitiven zu Programmieren.
Viel Spass!

3.3 Gemischte Variablen

Der Unterschied zu den Primitiven Variablen ist, dass man eine von Processing vorgegebene Klasse instanziert und deren Methoden erbt. Um jetzt nicht so weit auszuschweifen, will ich damit sagen, es verlangt eine andere Herangehensweise und Umgang mit ihnen .
Man beachte: Da der Typ String eine Instanz der Klasse String ist, wird er zu Anfang gross geschrieben - im Gegensatz zu den primitiven Variablen.

Wir wollen ein Zeichenkette wie „Hallo Welt" abbilden. Diese Zeichenkette besteht eigentlich aus zehn verschiedenen chars (die einzelnen Buchstaben). Es wäre doch mühsam alle zehn verschiedenen chars händisch zu einer Zeichenkette zu addieren! Dafür gibt es doch die String Klasse- sie macht alles viel einfacher.

° Strings (Zeichenketten):

Um einen String darzustellen gibt es mehrere Möglichkeiten.

```
String zeichenKette_1 = "Hallo Welt";

String zeichenKette_2 = new String("Hallo Welt 2");
```

oder auf die mühsame Art über chars.

```
char[] zeichenKette_3 = {'h','a','l','l','o'};
String zusammenGesetzt= new String(zeichenKette_3);
```

Man kann auch Zeichenketten aneinader hängen

```
String zusammenGesetzt= new String(zeichenKette_3);
```

Code:

```
String zeichenKette_1 = "Hallo Welt";
String zeichenKette_2 = new String("Hallo Welt 2");

char[] zeichenKette_3 = {'h','a','l','l','o'};
String zusammenGesetzt= new String(zeichenKette_3);

// Ausgabe
println(zeichenKette_1);
println(zeichenKette_2);
println(zusammenGesetzt);
```

Die Methoden der String Klasse

`length()` Gibt die Anzahl der Buchstaben im verwendeten String.
`charAt(int)` Gibt anhand dem Index einen Buchstaben zurück.
`indexOf(char)` Gibt den Index des Buchstaben im String zurück.
`equals(String)` Vergleicht zwei Strings miteinander. Gibt bool.
`toLowerCase(String)` Konvertiert String nach Kleinbuchstaben.
`toUpperCase(String)` Konvertiert String nach Grossbuchstaben.
`substring(int)` Liefert einen Teil des Strings als neuen String.

Ich möchte hier näher auf die einzelnen Methoden eingehen. Sie sind desweiteren alle in der Processing Dokumentation enthalten und beschrieben.

`- length()`
Anders als bei den Primitven gibt uns die String Klasse Methoden vor um bestimmte Eigenschaften auszulesen. Man möchte also z.B. wissen, aus wie vielen Zeichen die Zeichenkette besteht, nutzt man die Methode `.length()`. Wenn sie aufgerufen wird, liefert sie einen Integer zurück, der weiss wieviele Buchstaben der String enthält. Schauen wir uns das in der Praxis an:

Code:

```
String zeichenKette_1 = "Hallo Welt";
println("Zeichenkette enhält: ");
println( zeichenKette_1.length() );    //- > 10
println("Buchstaben");
```

Wie wir in diesem Beispiel sehen können, ist es uns erlaubt, da der String ein Objekt ist - eine Methode darauf anzuwenden. Dies geschieht mit einem Punkt zwischen des Objekts und der Methode. -> `zeichenKette_1.length()`
Dieses Wissen können wir jetzt versuchen auf alle weiteren Methoden zu applizieren.

`- charAt(int)`
Diese Methode soll uns ein Buchstaben einer bestimmten Position des Strings zurück liefern. Wenn ich also den zweiten Buchstaben haben möchte benutze ich `String.charAt(1)`. In der Praxis:

Code:

```
String zeichenKette_1 = "Hallo Welt";
println( zeichenKette_1.charAt(1) ); // 'a'
```

- `indexOf(char)`

Wenn ich wissen möchte ob und an welcher Stelle ein bestimmtes oder mehrere Zeichen in einem String vorkommen oder nicht, bietet mir diese Methode das richtige Werkzeug. Falls das Gesuchte nicht im String gefunden werden kann liefert es -1 zurück. So kann man recht komfortabel überprüfen, ob z.B bestimmte Namen in Listen vorhanden sind.

Code:

```
// Prüfe ob der gesuchte Vorname enthalten ist
String myStr = "Wartmann : Christoph ";
println( myStr.indexOf("Christoph") ); // gibt 11
```

- `equals(String)`

Prüft auf Gleichheit zweier Strings und gibt einen boolschen Wert zurück.

Code:

```
// Prüfe ob der gesuchte Vorname enthalten ist
String myStr   = "Katze";
String suchStr = "Hund";
boolean pruefergebnis = myStr.equals(suchStr);
println( pruefergebnis ); // ergibt false
```

- `toLowerCase()`

Um einen String in Kleinbuchstaben zu konvertieren. Das Ergebnis kann einfach die Variable überschreiben.

Code:

```
// Konvertiere in Kleinbuchstaben
String myStr = "ChRiStoP WaRtmAnN";
myStr        = myStr.toLowerCase();
println( myStr ); // ergibt "christoph wartmann"
```

- `toUpperCase()`

Funktioniert nach dem gleichen Prinzip wie `toLowerCase()`.Mit dem Unterschied dass der String in Grossbuchstaben gewandelt wird.

Code:

```
// Konvertiere in Grossbuchstaben
String myStr = "ChRiStoP WaRtmAnN";
myStr        = myStr.toUpperCase();
println( myStr ); // ergibt "CHRISTOPH WARTMANN"
```

- substring(int)

Mit dieser Methode können wir einen Teil eines Strings kopieren. Dabei sollten wir aufpassen, denn diese Methode liefert einen neuen String zurück. Als Argument können wir den Anfangs- und Endpunkt festlegen.

Code:

```
// Schneide die letzten 5 Buchstaben aus dem String
String teilStr;     // Hier wird das Ergebnis gespeichert
String myStr = "Gedanken Mobil"; // Orginal String

// Scheidprozess
teilStr =
myStr.substring(myStr.length()-5,myStr.length());

// Ausgabe
println( teilStr ); // ergibt "Mobil"
```

° Arrays (Listen):

Es bleibt noch ein wenig trocken. Aber ich versuche den Code möglichst klein zu halten damit man das Kernproblem besser sehen und verstehen kann. Kommen wir zu den Arrays. Ein Array ist eine Art Schrank welcher mehrere Schubladen hat um Informationen rein zu stecken. Es kann aber auch so gross sein wie ein Hochhaus mit vielen Stockwerken. In jedem Stockwerk sind mehrere Zimmer in denen Schränke stehen usw. Da haben viele, viele Informationen Platz. Das Array ist also eine Art Container. Die Konstruktor des Arrays sieht wie folgt aus.

```
Typ[] name = new Typ[ Anzahl der Felder ];
```

Wie wir schon bei den „Strings" gesehen haben, brauchen wir eine Instanz von Array, die man durch „new" bekommt.

Ein Array in Processing kann immer nur einen Typ von Variable aufnehmen. Bei einer Namensliste wäre es der Typ String. Die Deklaration des Arrays sieht wie folgt aus.

```
String[] meinStrArray = new String[10];
```

Wir versuchen uns erstmal an einer Namensliste. Da Namen als Zeichenketten abgespeichert werden, ist unser Array vom Typ String. Eine Möglichkeit ein Array zu definieren, ist es zuerst zu deklarieren, um später Werte hinein zu schreiben. Die Anzahl der Felder des Arrays müssen schon am Anfang festlegt werden. Diese Anzahl kann in Processing im Nachhinein leider nicht mehr verändert werden. Der Index des ersten Felds fängt immer bei 0 an und nicht bei 1, wie man es ja eigentlich gewohnt ist. Das könne dann so aussehen:

Code:

```
// Deklariere Array vom Typ String
// mit 5 Feldern (0-4)
String[] namensListe = new String[5];

// Weise einen Namen einer Position zu
namensListe[0] = "Christoph"; // erste position
namensListe[1] = "Maria";     // zweite position
namensListe[2] = "Julia";     // etc...
namensListe[3] = "Martin";
namensListe[4] = "Josef";

// Zeige alle Elemente des Arrays
println( namensListe );

// Zeige ein Element des Arrays, z.B. das 2. Feld
println( namensListe[1] ); // zeigt 'maria'
```

Um an einzelne Werte des Array zu kommen schreibt man einfach den Index in eckige Klammern. So lassen sich auch Werte überschreiben.

```
array[index] = neuerWert.
```

Das war jetzt ein ein-dimensionales Array. Als nächstes möchte ich auch gern auf ein zwei-dimensionales Array zu sprechen kommen. Im letzten Beispiel hatten wir nur verschieden Spalten. Jetzt kommen noch Zeilen hinzu, wie in einem Exel Sheet oder einer Tabelle. In der Ersten Spalte steht immer noch der Vorname. In der Zweiten steht der Nachname und in einer dritten Spalte der Beruf.

Tabellenstruktur:

Christoph	Maria	Julia	Martin
Wartmann	Seiler	Kromer	Kazula
Assistent	Hilfsassistent	Administrator	Student

Versuchen wir das mal ins Array zu übertragen.
Der Konstuktor eines 2D-Arrays enthält lediglich eine zweite eckige Klammer.
`String[][] namensListe2D .`
Um später an den jeweilen Wert zu kommen, müssen wir den Index der Spalte und der Zeile angeben. Schauen wir uns das an einem Beispiel an:

Code:

```
// Deklariere 2D Array vom Typ String
// mit 4 Feldern (0-4)
// In Zeile 0 steht der Vorname in den Spalten
// In Zeile 1 steht der Nachname in den Spalten
// In Zeile 3 steht der Beruf in den Spalten
String[][] namensListe2D = new String[3][4];

//Weise der 1 Zeile die Vornamen zu
namensListe2D[0][0] = "Christoph";
namensListe2D[0][1] = "Maria";
namensListe2D[0][2] = "Julia";
namensListe2D[0][3] = "Martin";

//Weise der 2 Zeile den Nachnamen zu
namensListe2D[1][0] = "Wartmann";
namensListe2D[1][1] = "Seiler";
namensListe2D[1][2] = "Kromer";
namensListe2D[1][3] = "Kazula";

//Weise der 3 Zeile den Beruf zu
namensListe2D[2][0] = "Assistent";
namensListe2D[2][1] = "Hilfsassistent";
namensListe2D[2][2] = "Administrator";
namensListe2D[2][3] = "Student";

// Output des ersten Eintrags der Tabelle
println("Erster Vor-Nachname: "+namensListe2D[0][0]+" "
+namensListe2D[1][0]+" "+namensListe2D[2][0]);
```

Das Array hat eine `length` Eigenschaft. Sie gibt die Anzahl der Felder zurück.

Für die Anzahl der Zeilen:

```
println("Anzahl der Zeilen: "+namensListe2D.length);
```

Für die Anzahl der Spalten:

```
println(" Anzahl der Zellen: "+namensListe2D[0].length);
```

Um eine sinnvolle Einsatzmöglichkeit zu zeigen, möchte ich das gesamte Array, mit allen Feldern, Auslesen und Anzeigen lassen. Dies lässt sich am besten mit einer for-Scheife realisieren, eigentlich zwei. Man müsste sonst das Array händisch, Zeile für Zeile auslesen; wäre mühsam und nicht besonders dynamisch, da man bei Grössenänderungen des Arrays, Grossteile des Codes immer wieder schreiben würde.

Code:

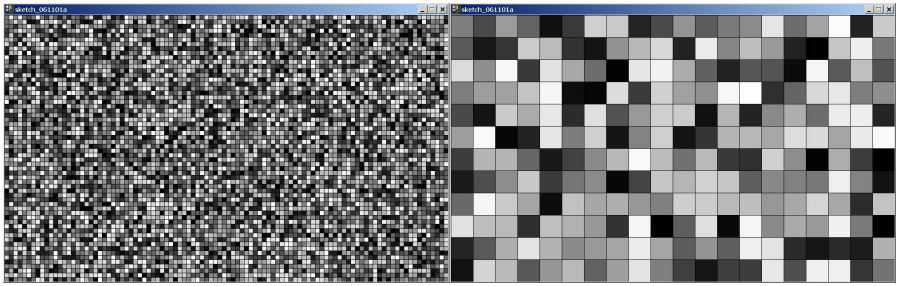
```
// Schleift durch die Zeilen
for(int i = 0; i < namensListe2D.length; i++)
{
  // Output Zeile
  println("Zeile: "+i);
  println("-----------------------------------------------");

  // Schleift durch die einzelnen Zellen
  for(int j = 0; j < namensListe2D[0].length; j++)
  {
    // Schreibt die Inhalte der Zellen in den Output
    println(" Zelle:"+j+" Inhalt: "+namensListe2D[i][j] );
  }

  println("-----------------------------------------------");
}
```

Die erste Schleife geht durch alle Zeilen des Arrays bis „length" erreicht ist. Bei jedem Durchgang 0..1..2.. usw. führt es eine zweite Scheife aus, die wiederum alle weiteren Felder ausliesst bis lenght erreicht ist. Dann fängt es wieder von vorne an. Diese Technik werden wir im weiteren Verlauf immer wieder antreffen, deshalb lohnt es sich jetzt schon sie einer genaueren Studie zu unterziehen.

Übung zum Kapitel:
Also, gibt es hier noch eine nette Übung mit grafischer Ausgabe dazu um das Wissen zu vertiefen. Und so könnte es aussehen:



Die Übung besteht darin, Quadrate einer varablen Grösse und Farbe auf einer variablen Appletfläche, ganzflächig zeichnen zu lassen.

Vom Konzept her könnte es so gehen:
Wir definieren als erstes die Bühnengrösse, dann bestimmen wir die Grösse des Quadrats. Daraus ergibt sich die Anzahl der Quadrate, die wir zum Füllen der Bühne brauchen *(Bühnengrösse / Quadratgrösse)*. Die erste for-Schleife regelt den vertikalen Vorschub *(i * Höhe)* solange die Maximalhöhe nicht erreicht ist. Die Zweite den Horizontalen (*j* Breite*). Vor dem Zeichnen, generieren wir uns eine Zufallsfarbe, hier einen Grauwert über *random(int).* Das Zeichnen von Quadraten folgt den gleichen Regeln wie das Zeichenen von Ellipsen.

Kopier doch den Code - spiel mit den Variablen der Bühnengrösse (size) und der Quadratgrösse. Du wirst sehen, die Rechtecke werden sich automatisch an den Rand des Applets anpassen. Falls dir das Malen eines Rechtecks noch ein Rätsel ist, schau doch geschwind in die Dokumentation und mache ein eigenes Beispiel dazu.

Code:

```
size(800,600);                  // Appletgrösse
int quadgroesse = 50;           // Grösse des einzelen Quadrats
int zeilen = height/quadgroesse;// Zeilen Berechnung
int zellen = width/quadgroesse; // Zellen Bereichnung

for(int i = 0; i < zeilen; i++)
{
  for(int j = 0; j < zellen; j++)
  {
     fill(random(0,255));        // Setzen einer Zufallsfarbe
     //Und Loszeichenen
     rect((j*quadgroesse),(i*quadgroesse),quadgroesse,quadgroesse);
  }
}
```