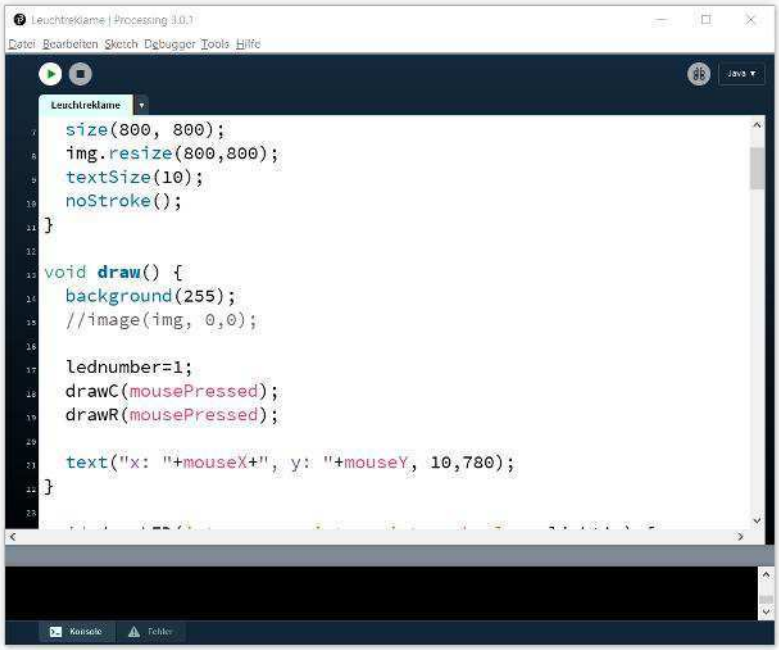
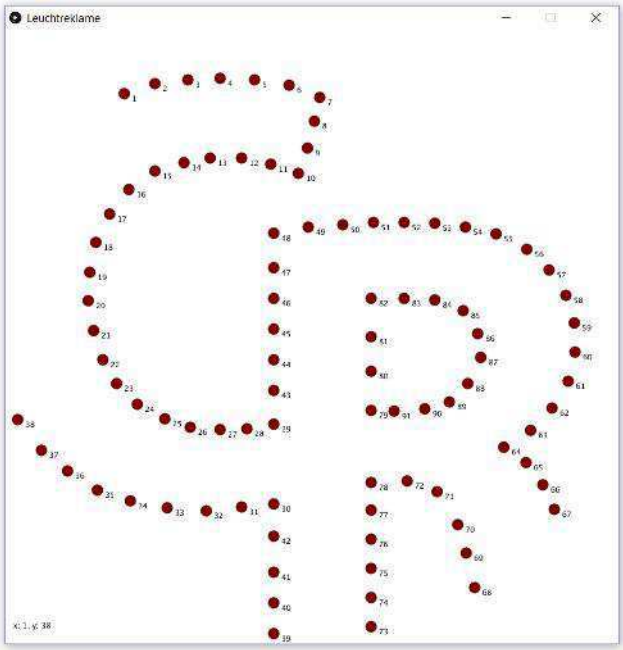




Introduction to Processing

After processing the following learning modules, you will be able to handle small programming projects.

The programming is done with the programming environment Processing, which can be downloaded free of charge from www.processing.org. Processing makes it very easy to realize graphical programs.

The upper secondary school uses this development environment to get first experiences with programming.

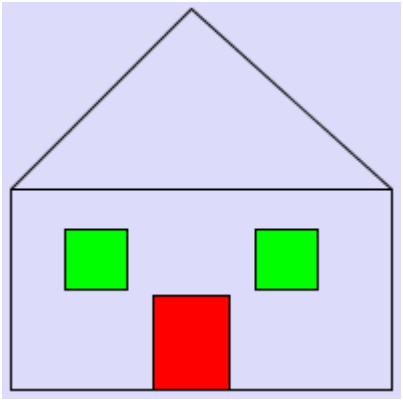
Drawing Graphics

In programming, the computer will be given instructions with a number of commands. These commands are called functions.

In the following, some features are introduced that generate graphics on the artboard of Processing.

working tasks – drawing graphics

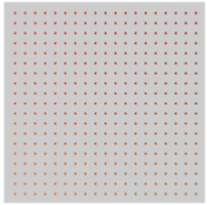
1. Make use of line and rect-drawing commands to design a colored house like the following.



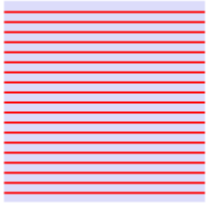
2. Develop a program that draws a smiley. For drawing circles you need to use a command called ellipse. Make use of the Processing reference documentation.

working tasks – drawing patterns

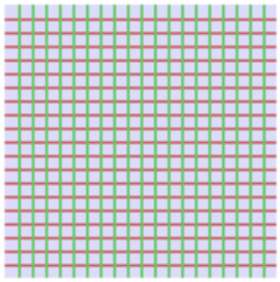
1. Create a large colored cloud made of points



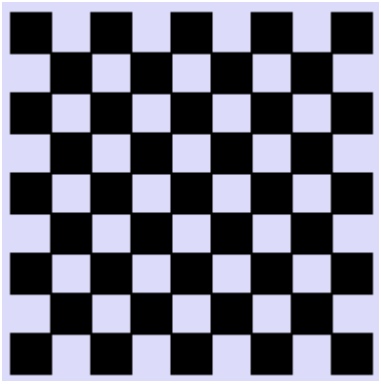
2. Create a pattern of red horizontal lines



3. Create a net consisting of red horizontal and green vertical lines

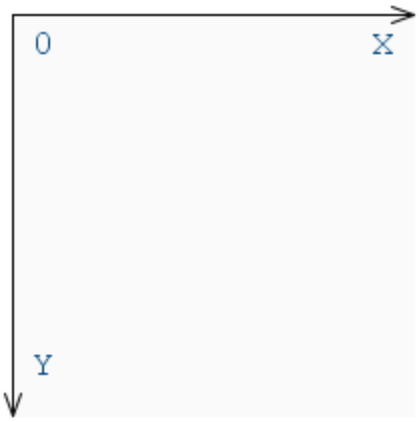


4. Create a chess field pattern



information

The coordinate system



Note that the coordinate system in the processing programming is defined as shown opposite.
As usual, the x-axis shows higher values on the right. The y-axis, on the other hand, is exactly the opposite as usual. Higher y values are positioned lower on the scale!

Drawing points

Points can be generated with the command point.

The coordinates will be established relatively from the upper left corner of the output window.

point

Syntax	point(x, y)
Parameter	x: x-coordinate of the point y: y-coordinate of the point

point

Example

```
point(10, 10); point(10, 12); point(10, 14);
point(10, 16); point(10, 18); point(10, 20);
```

```
point(12, 10); point(12, 12); point(12, 14);
```

```
point(12, 16);
point(12, 18); point(12, 20);point(14, 10); point(14, 12); point(14,
14);point(14, 16); point(14, 18); point(14, 20);
```

2. Drawing lines

Lines are generated with the command `line`:

line

Syntax	line(x1, y1, x2, y2)
Parameter	x1: x-coordinate of the first point y1: y-coordinate of the first point x2: x-coordinate of the second point y2: y-coordinate of the second point

Example

```
line(10, 80, 30, 40);
```

```
line(20, 80, 40, 40);
line(30, 80, 50, 40);
line(40, 80, 60, 40);
line(50, 80, 70, 40);
```

3. Drawing rectangles

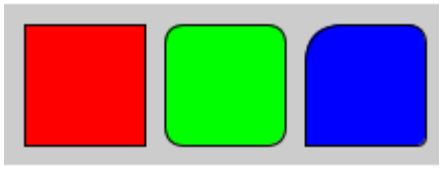
Rechtecke bestehen aus einem Rand und einer Füllung. Sie können mit dem rect-Befehl erzeugt werden.

rect

Syntax	rect(a,b,c,d) rect(a,b,c,d,r) rect(a, b, c, d, tl, tr, br, bl)
Parameter	a: x-coordinate b: y-coordinate c: width d: height r: radius for rounded corner tl: radius for top left corner tr: radius for top right corner br: radius for bottom right corner bl: radius for bottom left corner

Example

```
fill(255,0,0);
rect(10, 10, 60, 60);
fill(0,255,0);
rect(80, 10, 60, 60, 10);
fill(0,0,255);
rect(150, 10, 60, 60, 20, 10, 5, 0);
```



4. Changing the drawing area

You can change the drawing area with the commands size and background.

size

Syntax	size(w, h)
Parameter	w: width of the drawing area h: height of the drawing area
Description	allows to adjust the size of the drawing area

Syntax	background(rgb)

	`background(rgb, alpha)`
`Parameter`	`rgb: any color value` `alpha: opacity`
`Description`	`allows to colour the drawing area`

5. Representation of colours

Colours can be defined in different ways. In Processing the following representations are possible:

Description	**Beispiel**
`RGB-individual values` `All colour components can have values between 0 and 255`	`background(204, 153, 0)`
`Greyscale` `A grey tone is used with values between 0 and 255`	`background(155)`
`RGB-hex-representation` `Colour components are displayed in hexadecimal numbers`	`background(#FFCC00)`

In addition, opacity with values between 0 and 255 can be specified in many cases, to influence the visibility of the object background.

6. Fillings and stroke colour and stroke width

In order to change the representation of drawn objects you use the commands `stroke, strokeWeight` `and fill`.

stroke	
Syntax	stroke(rgb) stroke(rgb, alpha)

stroke

Parameter	rgb: any colour value alpha: opacity
Description	changes the colour of lines and borders for all the following commands.

strokeWeight

Syntax	strokeWeight(weight)
Parameter	weight: weight in pixels
Description	changes the weight of lines and borders for all the following commands.

fill

Syntax	fill(rgb) fill(rgb, alpha)
Parameter	rgb: any colour value alpha: opacity
Description	changes the colour of the filling for all the following commands.

information

Using variables

It is often helpful to use as few concrete numbers or texts as possible. An example illustrates the meaning and usefulness of variables.

Following are two program versions for drawing vertical line patterns:

Variante A

```
line(0, 0, 0, 100);
(1)
(2)     line(10, 0, 10, 100);
(3)     line(20, 0, 20, 100);
(4)     line(30, 0, 30, 100);
(5)     line(40, 0, 40, 100);
```

Variante B

```
int x;
(1)
(2)     int y1;
(3)     int y2;
(4)     int distance;
(5)     distance = 10;
(6)     x = 0;
(7)     y1 = 0;
(8)     y2 = 100;
(9)     line(x, y1, x, y2);
(10)    x = x + distance;
(11)    line(x, y1, x, y2);
(12)    x = x + distance;
(13)    line(x, y1, x, y2);
(14)    x = x + distance;
(15)    line(x, y1, x, y2);
(16)    x = x + distance;
(17)    line(x, y1, x, y2);
(18)    x = x + distance;
```

The variant A offers in very short form the possibility to create a line pattern. It has the major disadvantage that changes and subsequent adaptation are very difficult to implement. For example, the requirement to change the distance between the lines can only be met by adjusting each line of the code.

Variant B is much longer in the first moment, but allows a simple change of the pattern. Moreover, as will be shown later, the representation can be varied much more easily.

The basic element of variant B is the variable. Instead of using fixed values in the program code, variables are used. These variables can be used with almost any name in the program. For this they must first be declared.

1. Step: Declare a name for the variable

The instruction „`int x;`“ creates a integer variable with the name **`x`** .

2. Step: Initalizing the variable x with a value

Now it must be determined which numerical value hides behind a name. The line „**`x`** `= 0;`“ ensures that the name **`x`** stands for the numerical value 0 starting at this line. The variable **`x`** is assigned the value 0.

3. Step: Use of the variable

Now the variable can be used to For example, to draw a line:

```
line(x, y1, x, y2); // equals line(0, y1, 0, y2) if x is set to 0
```

Or to change the contents of the variable:

```
x = x + distance; // equals x = 0 + 20; → x = 20; if x is 0
```