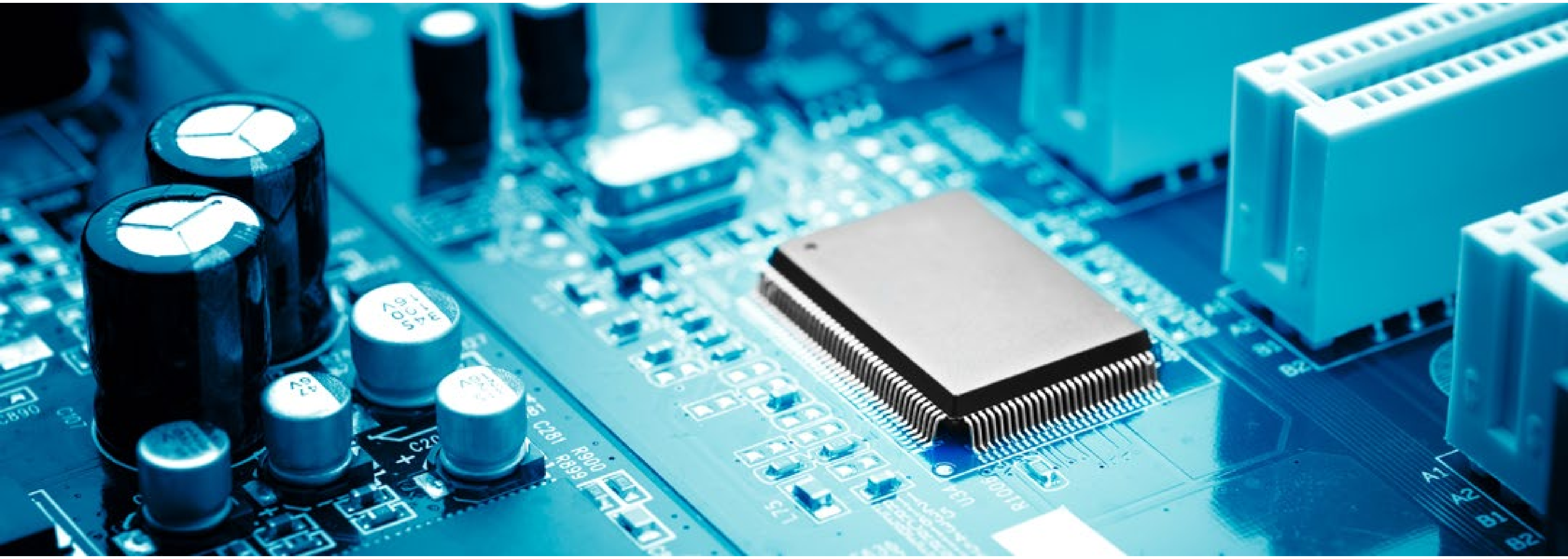
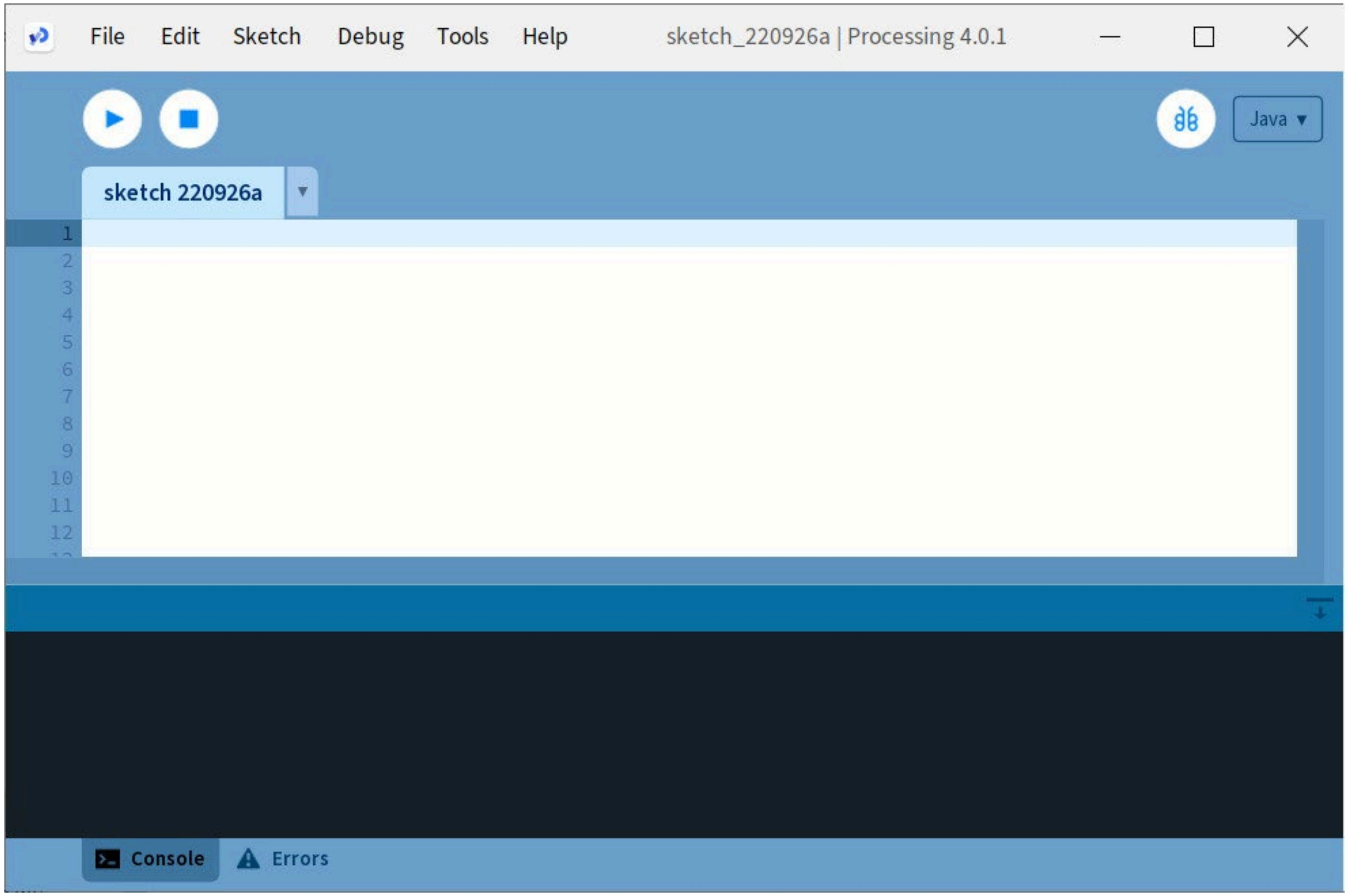


Processing – our first Program

The Processing IDE



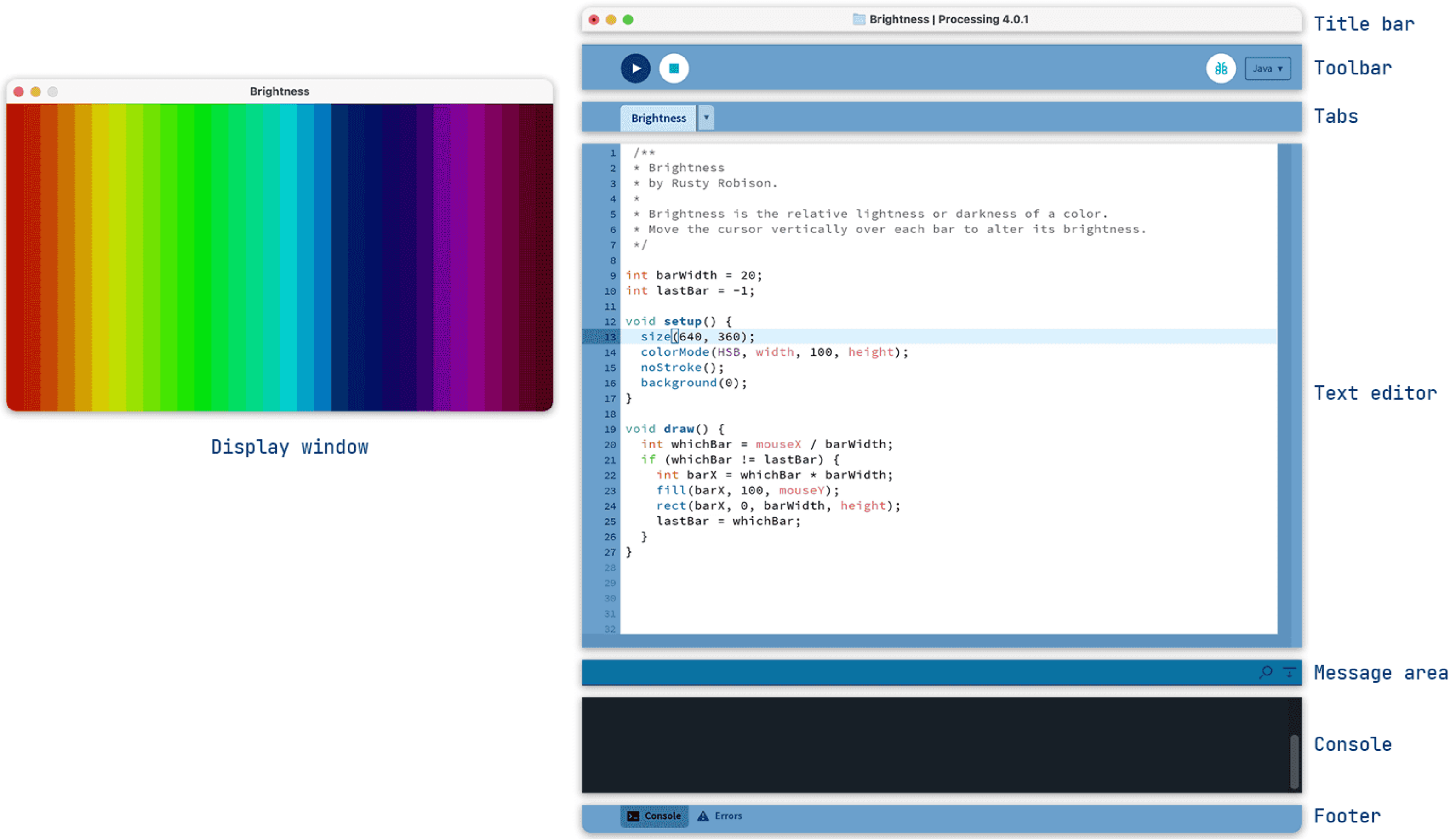
Screenshot of the Processing IDE main window

- Installation of Processing is pretty easy: https://processing.org/download
- Available for Windows/Mac/Linux

Processing Development Environment (PDE)

- The PDE consists of:
  - Simple text editor
  - Console
  - Other useful stuff…
- A program is called a „**sketch**“ (textfile with .pde ending).
- Sketches are stored in a “sketchbook” (directory on your harddisk.
- The **“run” button** compiles the code and opens the display window.

Processing Development Environment (PDE)



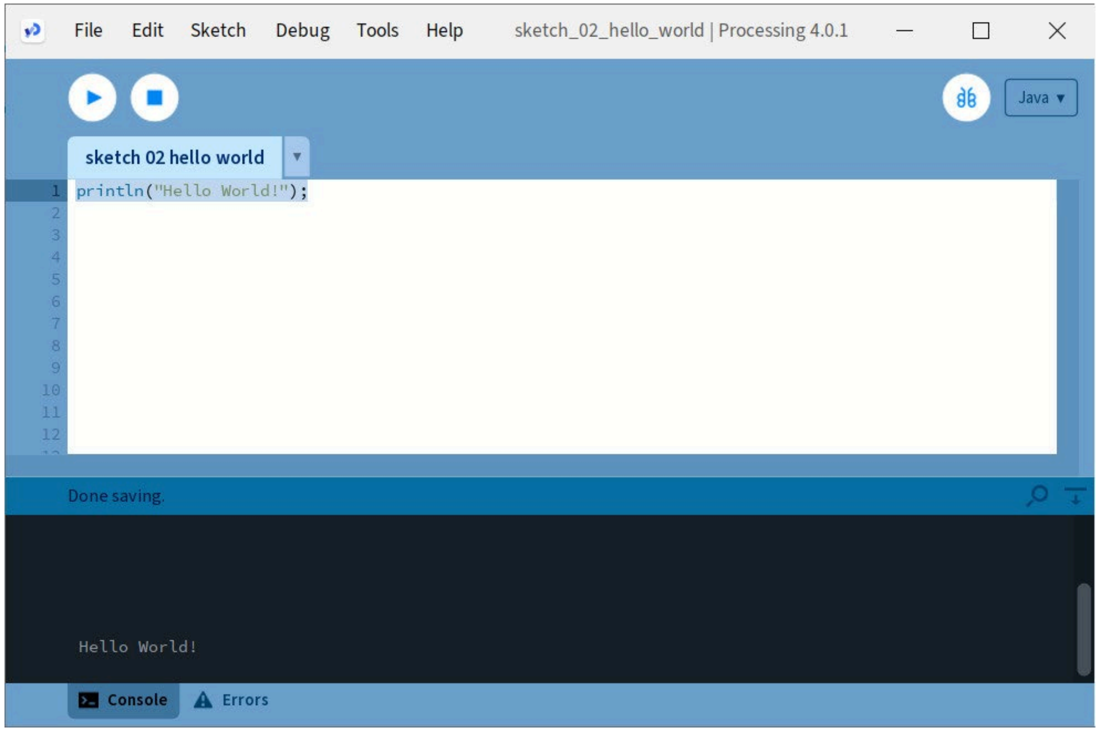
```
/**
 * Brightness
 * by Rusty Robison.
 *
 * Brightness is the relative lightness or darkness of a color.
 * Move the cursor vertically over each bar to alter its brightness.
 */

int barWidth = 20;
int lastBar = -1;

void setup() {
  size(640, 360);
  colorMode(HSB, width, 100, height);
  noStroke();
  background(0);
}

void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}
```

Components of the Processing PDE

From https://processing.org/environment/#processing-development-environment-pde on 2022-09-26,

Processing is distributed under CC BY-NC-SA 4.0

Our first program!



```
println("Hello World!");
```

Done saving.

```
Hello World!
```

Processing PDE: Screenshot of the „Hello World" sketch

```
println("Hello World!");
```

```
Hello World!
```

Our program code

Console output

Statements

```
println("Hello World!");
```

$\rightarrow$

```
Hello World!
```

- **A program consists of statements.**
  To be more precise: A sequence of statements.
  (That holds for procedural programming which we'll do. There are other forms of programming: Functional, object-oriented, declarative)
- Most of the times, in Processing/Java statements are ended with a `;`
- Statements can be:
  - **Functions**
  - Variable declarations
  - Assignments
  - Control structures…

Functions

```
println("Hello World!");
println("Hello again!!!");
```

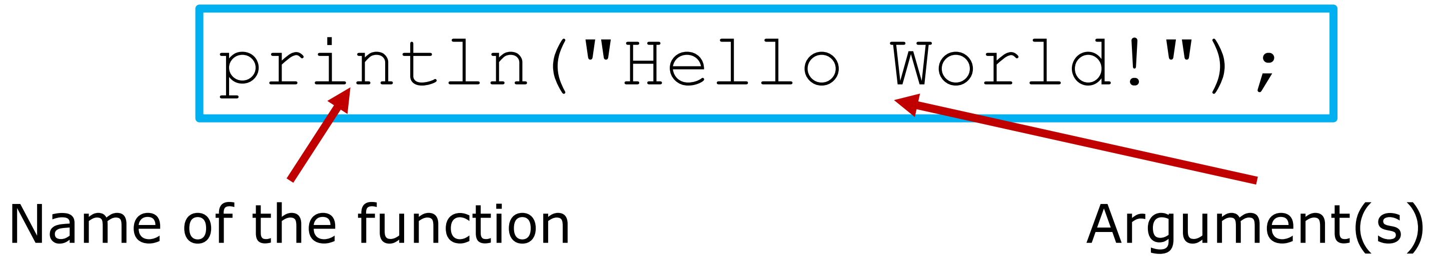
```
Hello World!
Hello Again!!!
```

Top-Down execution

- Statements are executed (unless otherwise defined by control structures) **line by line, top-down**.
- A **function** is
  - A **block of code** that can be executed.
  - It is **referenced by a name and an argument list**.
  - Processing has many **built-in functions**.
  - Has a return value.

Remark: The nature of learning / teaching programming is that sometimes concepts have to be introduced just roughly first, and detailed on later.

Predefined function usage



```
println("Hello World!");
```

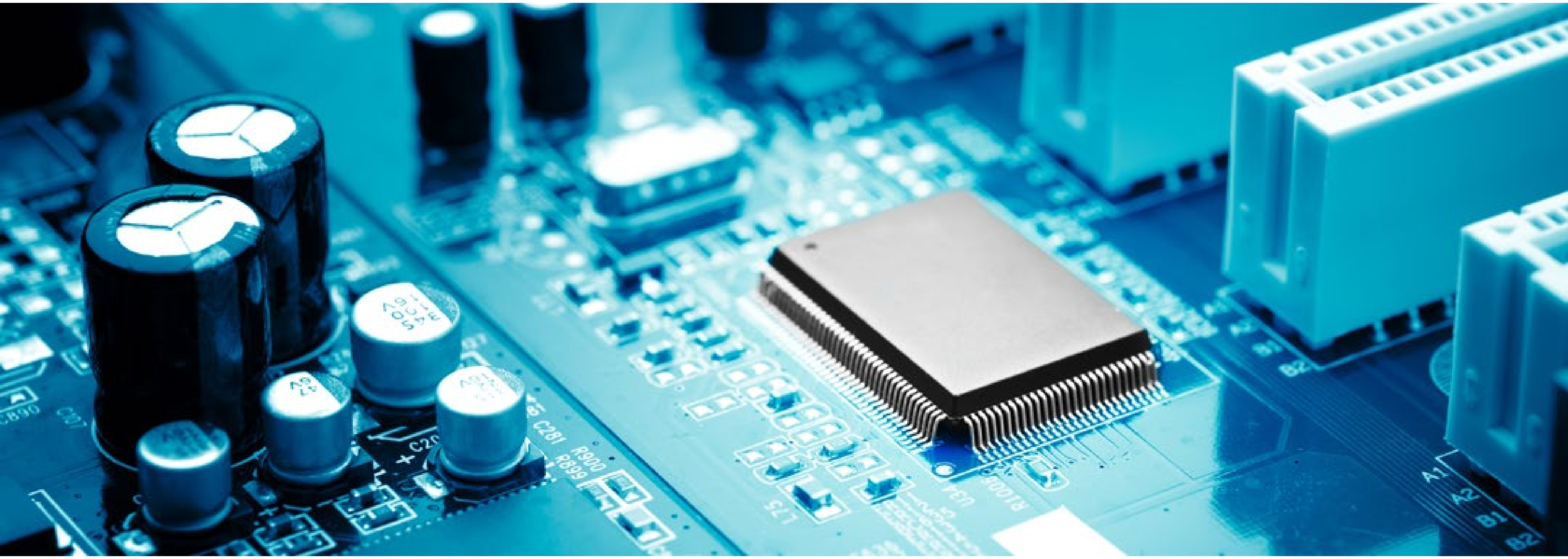
- The **Processing reference** is our friend when trying to find out what built-in functions are available: https://processing.org/reference/
- If we have sourcecode that uses a function unknown to us: Rightclick on it -> “Find in reference”

Printing to the console

```
println("This is a text");
println(24);
println(2.4);
println("Text and ", 32, " and ", 48);
print('a');
print('b');
println('c');
println("A new line");
```

```
This is a text
24
2.4
Text and  32  and  48
abc
A new line
```

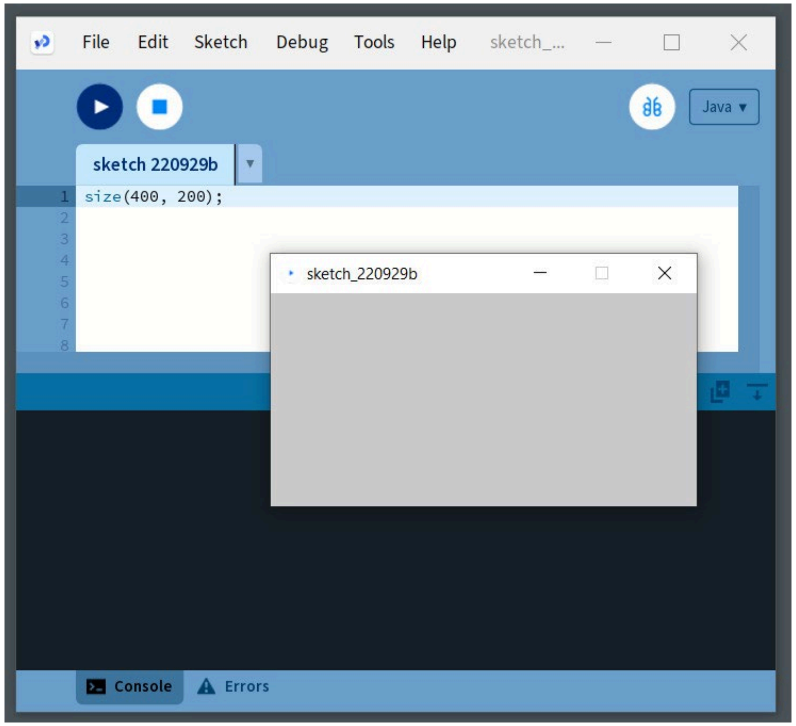
- There are 2 printing functions: `print` & `println`
- Both accept **characters, text and numbers** separated by commas **as arguments**



Digital images

The drawing area

- When you run a Processing sketch, a window opens.
- That is **an image** (drawing area)!
- You can **set its size** with the function
  `size(width, height);`
- The width and height is measured in "number of pixels".



```
size(400, 200);
```

Processing PDE: Screenshot of the drawing area

Image coordinate system

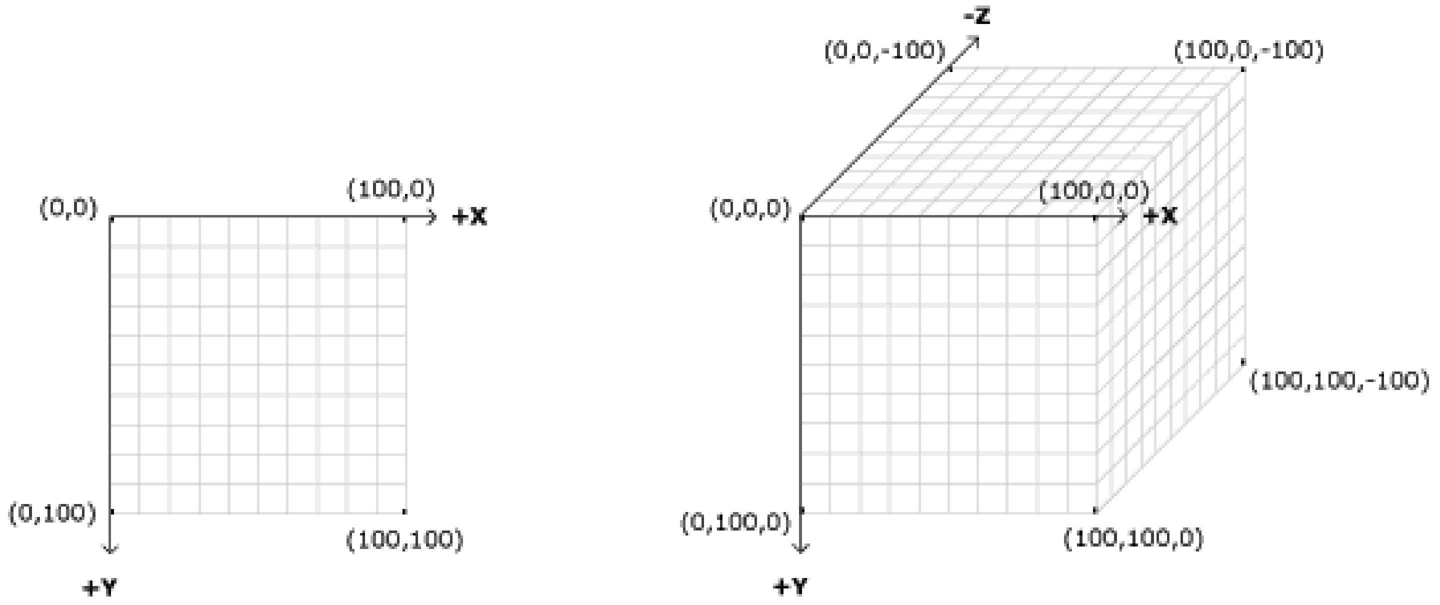


Image coordinate system

From https://processing.org/environment on 2022-09-29, Processing is distributed under CC BY-NC-SA 4.0

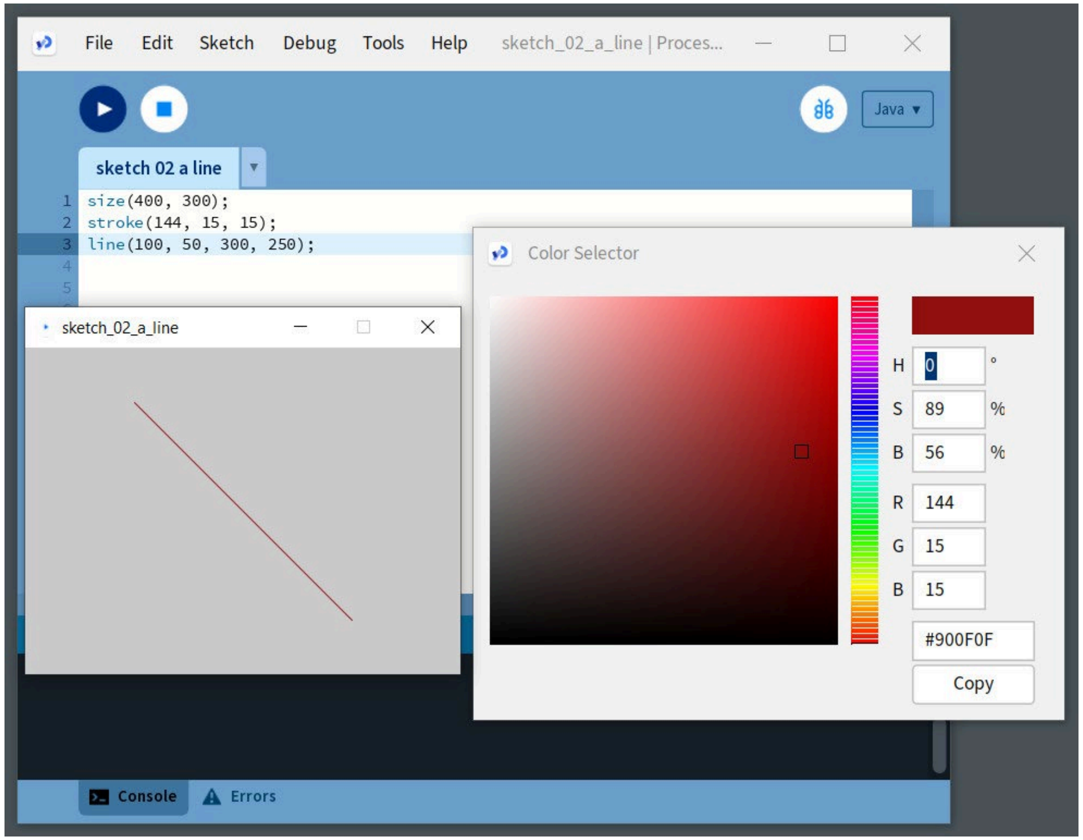
- **(0,0)** is the coordinate of the **top-left** pixel
- **(width – 1, height -1)** is the coordinate of the **bottom-right** pixel (pixels are drawn to the bottom right of the coordinate)
- There are no negative coordinate values

What is a pixel?

- A **digital image** is (in general) a **uniformly spaced grid** of pixels: Small squares with a certain color
- Pixels can be:
  - **Grayvalued**, i.e. the gray value is represented by a single value, i.e. in „basic" Processing: An integer in [0;255]. Low value (0) = black, high value (255) = white
  - **Colored**, i.e. the color values is represented by 3 values: Red, Green, Blue (in "basic" Processing: 3 integers in [0;255]).
  - Transparent (not considered as of now)
- To find a color: Menu item "Tools" → „Color Selector"

Drawing a line

- We can draw a **colored line** on to our drawing area with the help of two functions:
  - `stroke(…)`
  - `line(…)`



```
size(400, 300);
stroke(144, 15, 15);
line(100, 50, 300, 250);
```

Processing PDE: Screenshot of the color selector and a drawing of a line

DEGGENDORF INSTITUTE OF TECHNOLOGY

`stroke(…)` arguments

- `stroke(…)` **sets the color used to draw lines** and borders around shapes.
- Arguments can be (among others) a **gray value** (single number) **or an RGB color value** (3 numbers separated by comma)
- The color is used until the next call to stroke(…)
- Reference: https://processing.org/reference/stroke_.html

Set the line color to middle-gray

```
stroke(127);
stroke(0, 255, 0);
```

Set the line color to pure green

`line(…)` arguments

- `line(…)` **draws a line** defined by a start and an end coordinate
- 2D syntax: `line(x1, y1, x2, y2)`
  - `x1, y1:` Start coordinate
  - `x2, y2:` End coordinate
- The **line is colored** as defined by last `stroke(…)` call
- Reference: https://processing.org/reference/line_.html

```
line(100, 50, 300, 250);
```

The line starts at coordinate (100, 50)

The line ends at coordinate (300, 250)

`save(...)` : Storing images

- `save(filename)` **stores** the current drawing area as **an image**.
- The format will be determined by the filename ending.
- The image is stored in the sketch directory (File menu → Sketchbook → Show Folder).
- Reference: https://processing.org/reference/save_.html

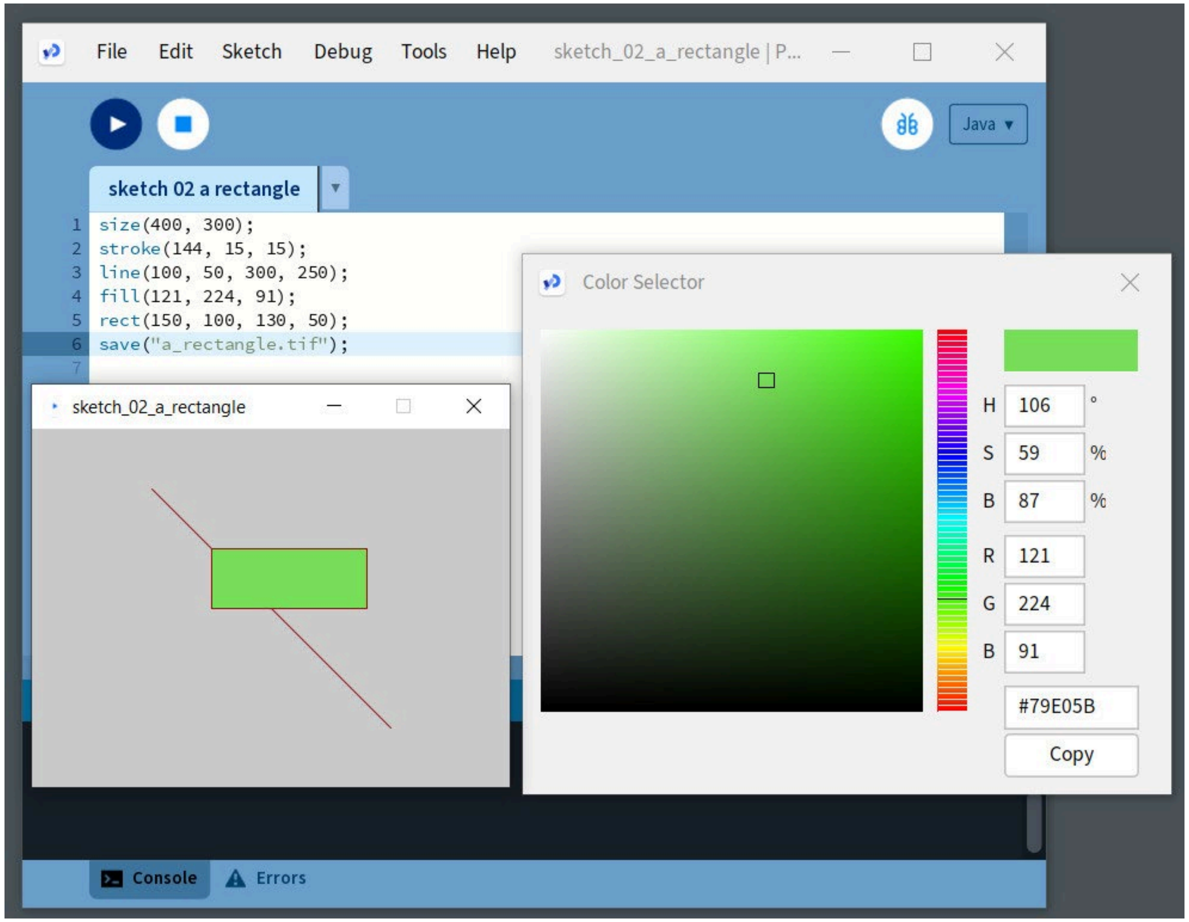
```
save("a_line.tif");
```

A TIF file with name „a_line.tif" will be created.

Stored image of the „a line" sketch shown before

Drawing a rectangle

- We can draw a **colored rectangle**:
  - `rect(…)`
  - `fill(…)`



```
size(400, 300);
stroke(144, 15, 15);
line(100, 50, 300, 250);
fill(121, 224, 91);
rect(150, 100, 130, 50);
save("a_rectangle.tif");
```

Processing PDE: Screenshot of the color selector and a drawing of a line and rectangle

DEGGENDORF INSTITUTE OF TECHNOLOGY

`fill(…)` and `rect(…)`

- `fill(…)` sets the **color used to fill shapes**
  - Possible arguments: See `stroke(…)`
  - The color is used until the next call to `fill(…)`
  - Reference: https://processing.org/reference/fill_.html
- `rect(…)` draws a **rectangle**
  - Basic argument syntax: `rect(x, y, width, height)`
  - Position (x,y), dimension (width, height)
  - The rectangle is (by default) drawn to the right-bottom of the position.
  - There are way more possibilities for arguments…
  - Reference: https://processing.org/reference/rect_.html

Other 2D drawing functions

- Color setting:
  - `background(…)`: Set the background color and erase previous drawings
  - `noStroke()` and `noFill()`: Do not print an outline/ filling
- 2D primitive shapes:
  - `circle(…)`
  - `ellipse(…)`
  - `triangle(…)`
- You‘ll find **detailed descriptions** with **examples** and references to similar functions in the Processing reference: https://processing.org/reference

Revision

- Read https://processing.org/tutorials/color until „RGB color"
- Read https://processing.org/tutorials/coordinatesystemandshapes
- Watch https://www.youtube.com/playlist?list=PLRqwX-V7Uu6ZYJC7L-r6rX6utt6wwJCyi (The Coding Train: Introduction to processing 0.0 – 0.6)
- Watch https://www.youtube.com/watch?v=a562vsSI2Po&list=PLRqwX-V7Uu6bsRnSEJ9tRn4V_XCGXovs4 (The Coding Train: Pixels)
- Watch https://www.youtube.com/playlist?list=PLRqwX-V7Uu6bsRnSEJ9tRn4V_XCGXovs4 (The Coding Train: Processing environment 2.1 and 2.2)

Remark: The **web tutorials** (written and video) have a **different structure then the lesson**. Sometimes they detail more on certain topics, sometimes they detail less. If a tutorial is written down as an exercise, I assume it to be known.

Related material

- We start with the exercise performance!
  - Exercise performance general rules
  - Exercise performance task 1
- Exercises `01_exercises_first_steps.pdf`