

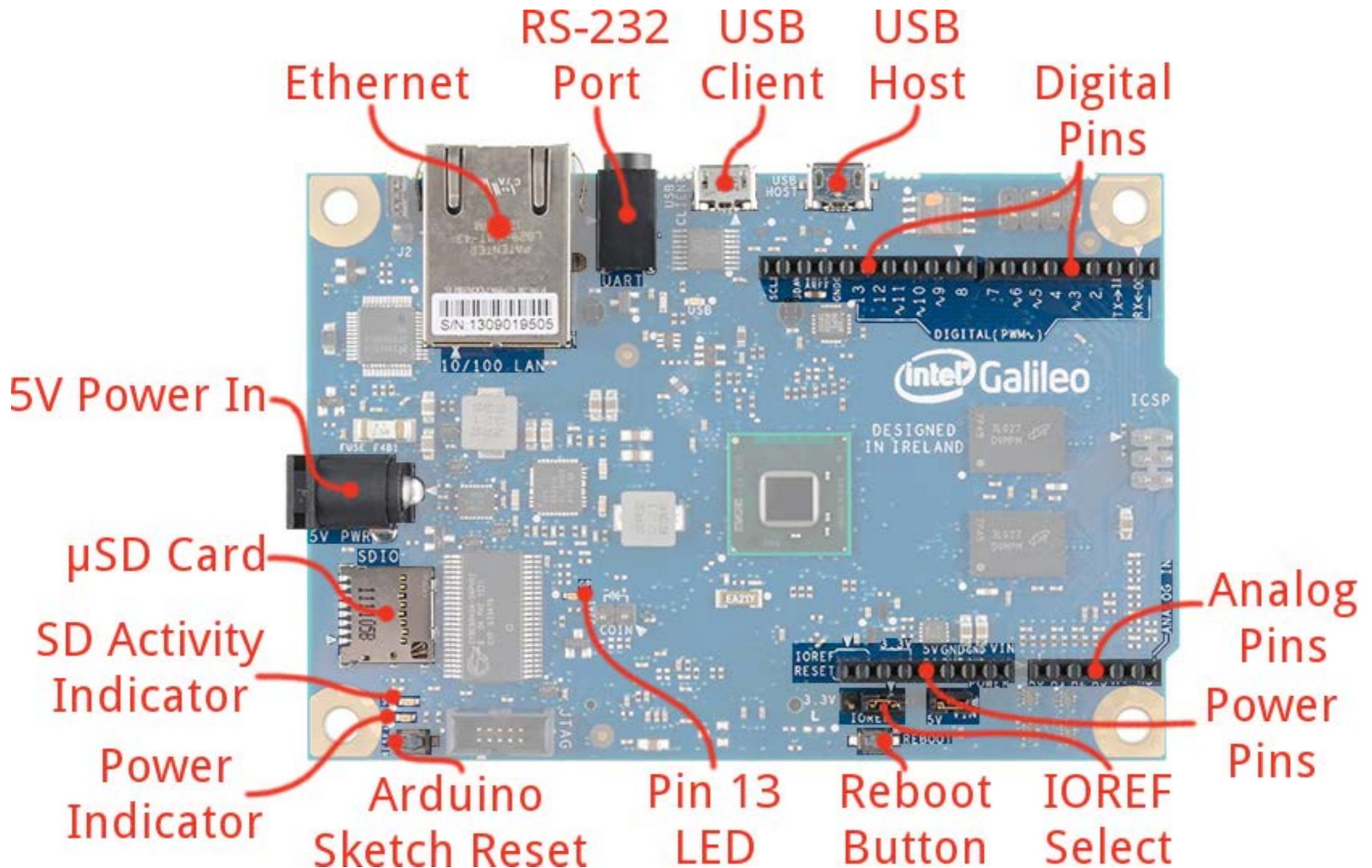


# Erste Schritte

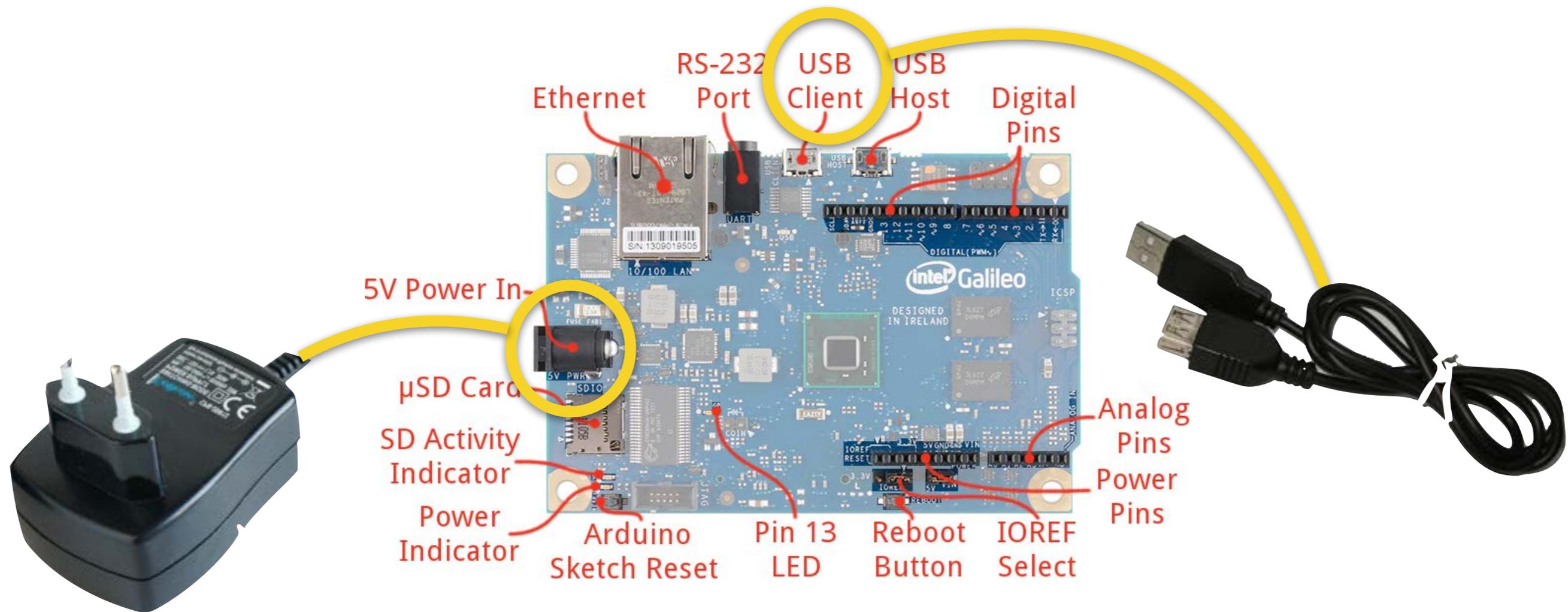
Programmieren für Ingenieure  
Sommer 2014

Andreas Zeller, Universität des Saarlandes

# Das Galileo-Board



# Anschluss



1. Netzteil

2. USB-Kabel

# Programmierungsumgebung

```
Blink | Arduino 1.5.3
Blink
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
  */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

– Download über Vorlesungs-Webseite –

Speichern abgeschlossen.

Transfer complete

#

#

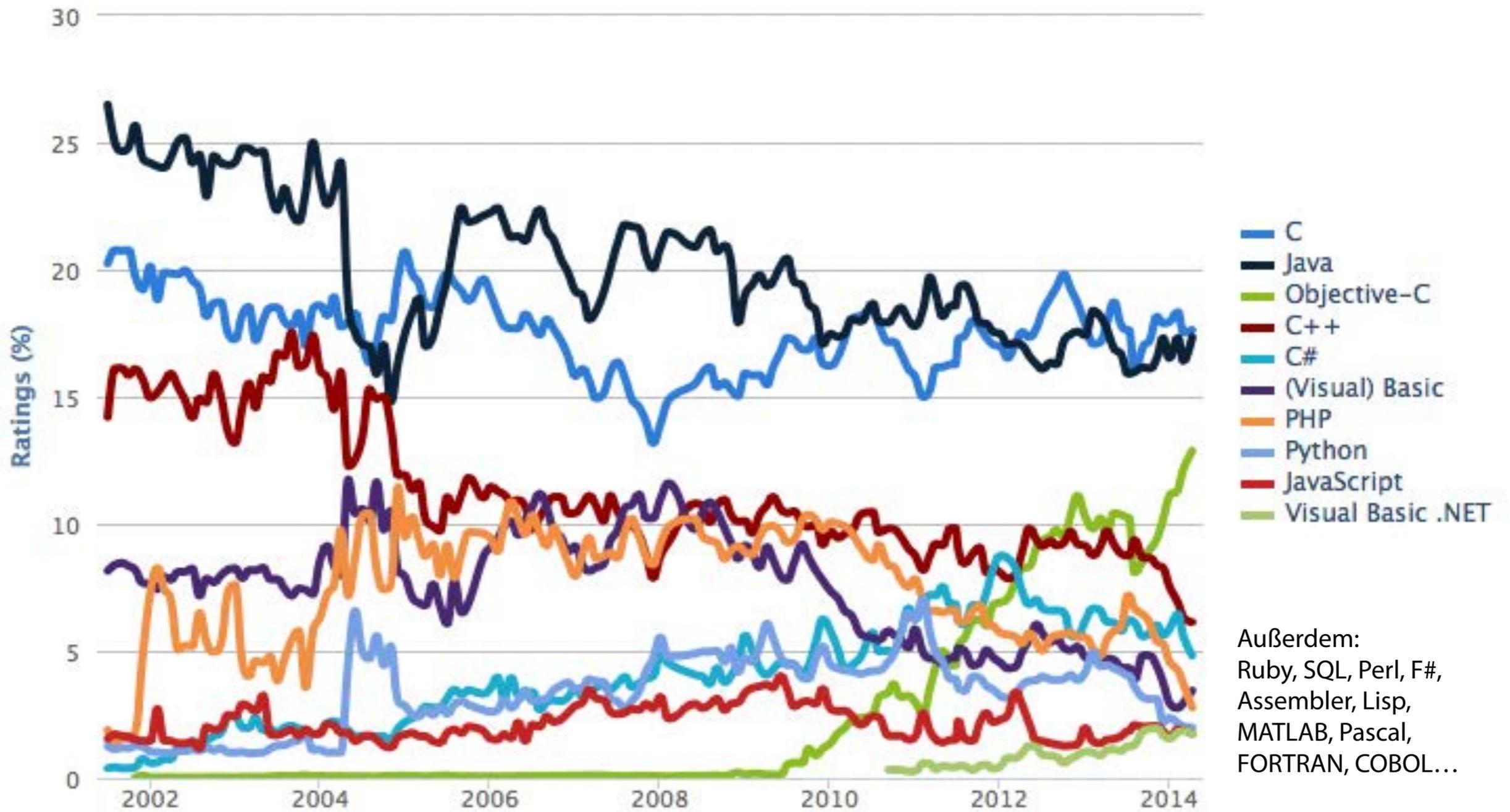
*Demo*

# Ein Programm

- Bestimmt, was der Rechner tun soll
- Geschrieben in einer *Programmiersprache*
- Enthält und organisiert *Anweisungen*

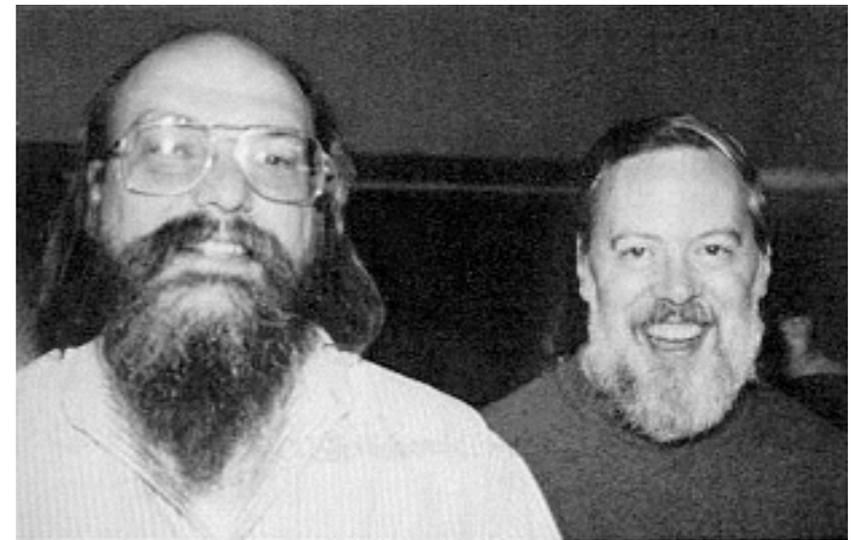
# Programmiersprachen

TIOBE Programming Community Index  
Source: [www.tiobe.com](http://www.tiobe.com)



# C

- Unsere Programmiersprache
- Entwickelt 1969–1973 in den Bell Labs für UNIX (als Nachfolger von B)
- Eine der verbreitetsten und einflussreichsten Sprachen
- Dialekte: C++, Objective-C



Ken Thompson und Dennis Ritchie,  
Erfinder der Programmiersprache C

# Ein C-Programm

- besteht aus *Anweisungen*:

```
digitalWrite(led, HIGH);
```

- die wiederum in *Funktionen* zusammengefasst werden:

```
void setup() {  
    pinMode(led, OUTPUT);  
}
```

- *Kommentare* erläutern den Zweck:

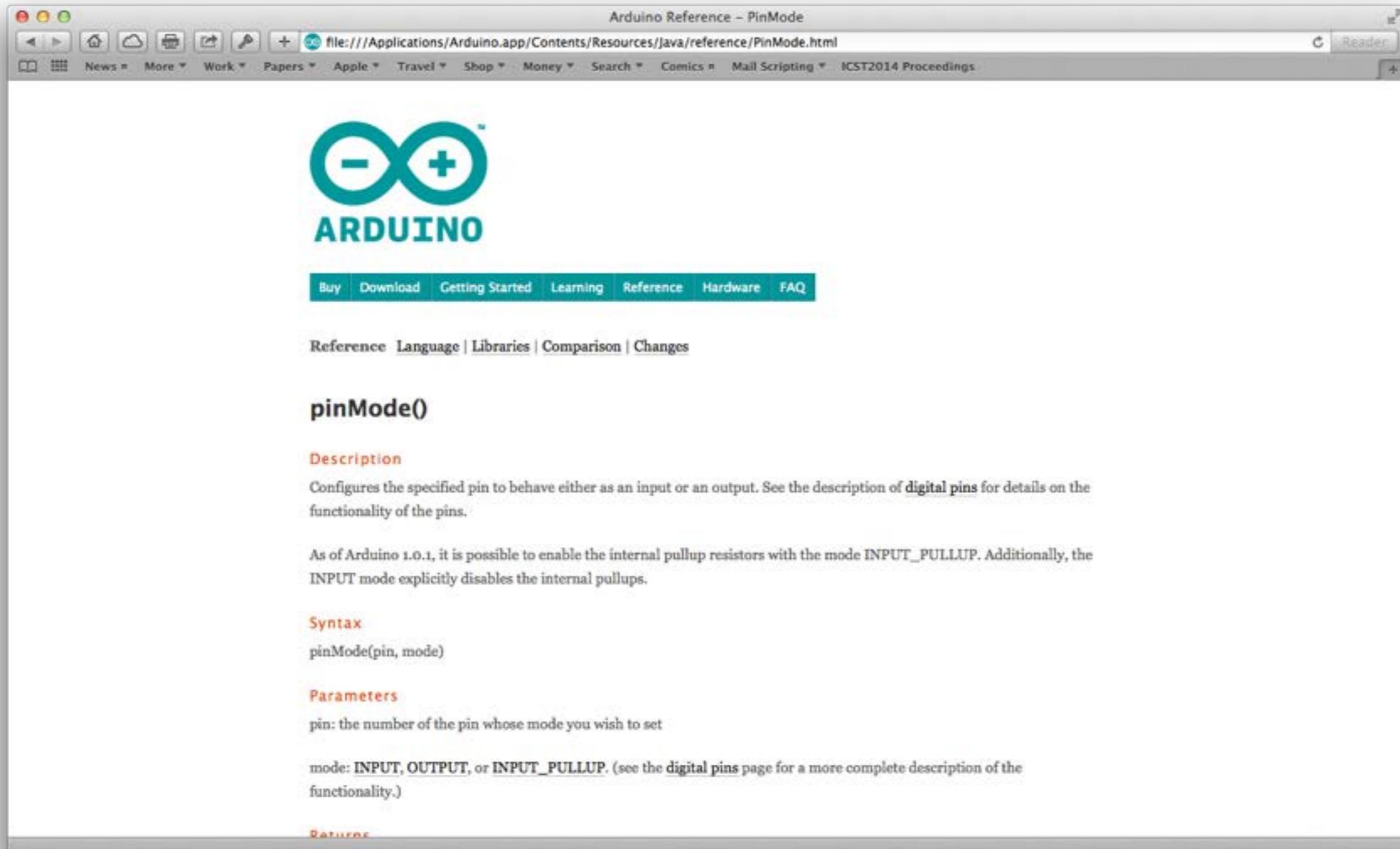
```
delay(1000);    // Eine Sekunde warten
```

# Anweisungen

- Wir betrachten zunächst *Funktionsaufrufe*.
- Die Arduino-Plattform stellt Tausende von *Funktionen* zur Verfügung
- Jede Funktion bietet einen *Dienst* an.

<code>pinMode()</code>	Pin als Ein/Ausgang konfigurieren
<code>digitalWrite()</code>	Daten digital ausgeben
<code>delay()</code>	Warten

# Alle Funktionen



The screenshot shows a web browser window titled "Arduino Reference - PinMode". The address bar displays the file path: `file:///Applications/Arduino.app/Contents/Resources/Java/reference/PinMode.html`. The browser's menu bar includes "News", "More", "Work", "Papers", "Apple", "Travel", "Shop", "Money", "Search", "Comics", "Mail Scripting", and "ICST2014 Proceedings".

The main content area features the Arduino logo (an infinity symbol with a minus sign on the left and a plus sign on the right) and the word "ARDUINO" below it. A navigation bar contains links for "Buy", "Download", "Getting Started", "Learning", "Reference", "Hardware", and "FAQ". Below this, there are links for "Reference", "Language", "Libraries", "Comparison", and "Changes".

## pinMode()

**Description**

Configures the specified pin to behave either as an input or an output. See the description of [digital pins](#) for details on the functionality of the pins.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode `INPUT_PULLUP`. Additionally, the `INPUT` mode explicitly disables the internal pullups.

**Syntax**

```
pinMode(pin, mode)
```

**Parameters**

`pin`: the number of the pin whose mode you wish to set

`mode`: `INPUT`, `OUTPUT`, or `INPUT_PULLUP`. (see the [digital pins](#) page for a more complete description of the functionality.)

**Returns**

In Arduino Menü: Hilfe → Referenz

# Funktionsaufrufe

- Die meisten Funktionen haben *Parameter*, die ihre Funktionsweise bestimmen

`digitalWrite(pin_number, value)`

- Beim Aufruf muss für jeden Parameter ein Wert (*Argument*) angegeben werden

`digitalWrite(13, HIGH);`

Funktionsname

Wert für *pin\_number*

Wert für *value*

# Vorgegebene Funktionen

- Jedes Arduino-Programm (*Sketch*) beginnt mit zwei Funktionen:

`setup()`

Einmalig beim Start ausführen

`loop()`

Immer wiederholen

- In diesen Funktionen wird festgelegt, was im Programm passieren soll.

# Funktionen definieren

- Eine Funktion wie `setup()` und `loop()` wird als Folge von Anweisungen definiert, eingeschlossen in {...}

```
void setup() {  
    Anweisung 1;  
    Anweisung 2;  
    ""  
}
```

- Jede Anweisung endet in einem ";"

# Kommentare

- *Kommentare* dienen dazu, das Programm für Menschen verständlich(er) zu machen
- Entweder `// ...` bis Zeilenende oder `/* ... */`

```
/* Pin 13 has an LED connected  
on most Arduino boards. */
```

```
// setup() runs once when you press reset
```

- Der Rechner *ignoriert* alle Kommentare

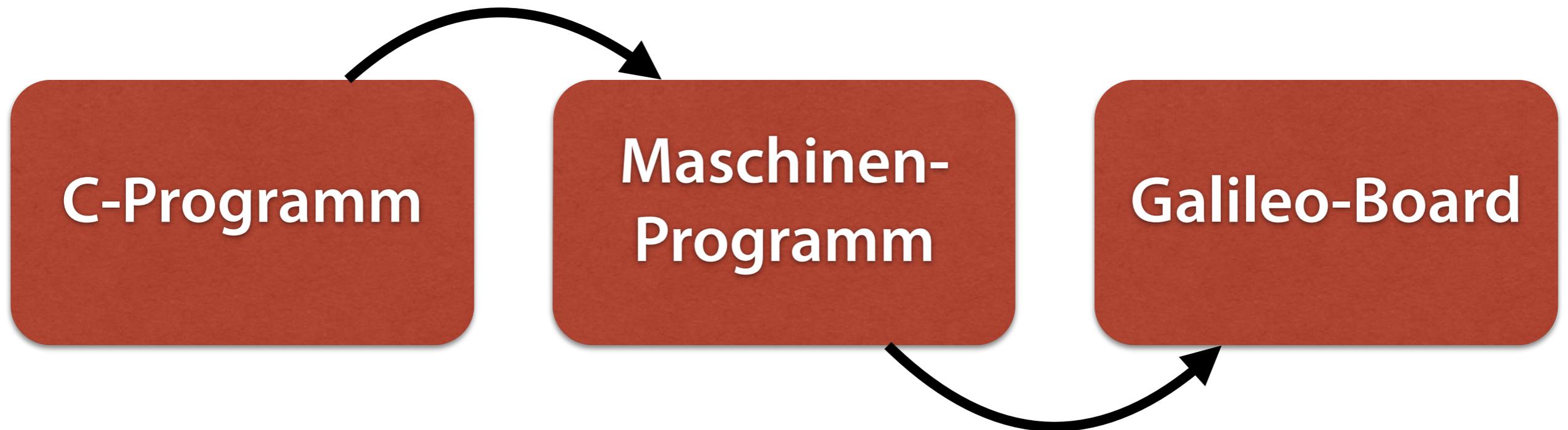
# Beispiel: 3x Blinken

```
void setup() {  
    // configure PIN 13 (built-in LED) as output  
    pinMode(13, OUTPUT);  
  
    // turn the LED on (HIGH is the voltage level)  
    digitalWrite(13, HIGH);  
  
    // wait for a second  
    delay(1000);  
  
    // turn the LED off by making the voltage LOW  
    digitalWrite(13, LOW);  
  
    // wait for a second  
    delay(1000);  
  
    // turn the LED on  
    ...  
}
```

*Demo*

# Vom Programm zum Prozessor

Prüfen und Übersetzen



Hochladen über USB



*Demo*

# Wiederholung

- Nach `setup()` wird die `loop()`-Funktion immer und immer wieder aufgerufen

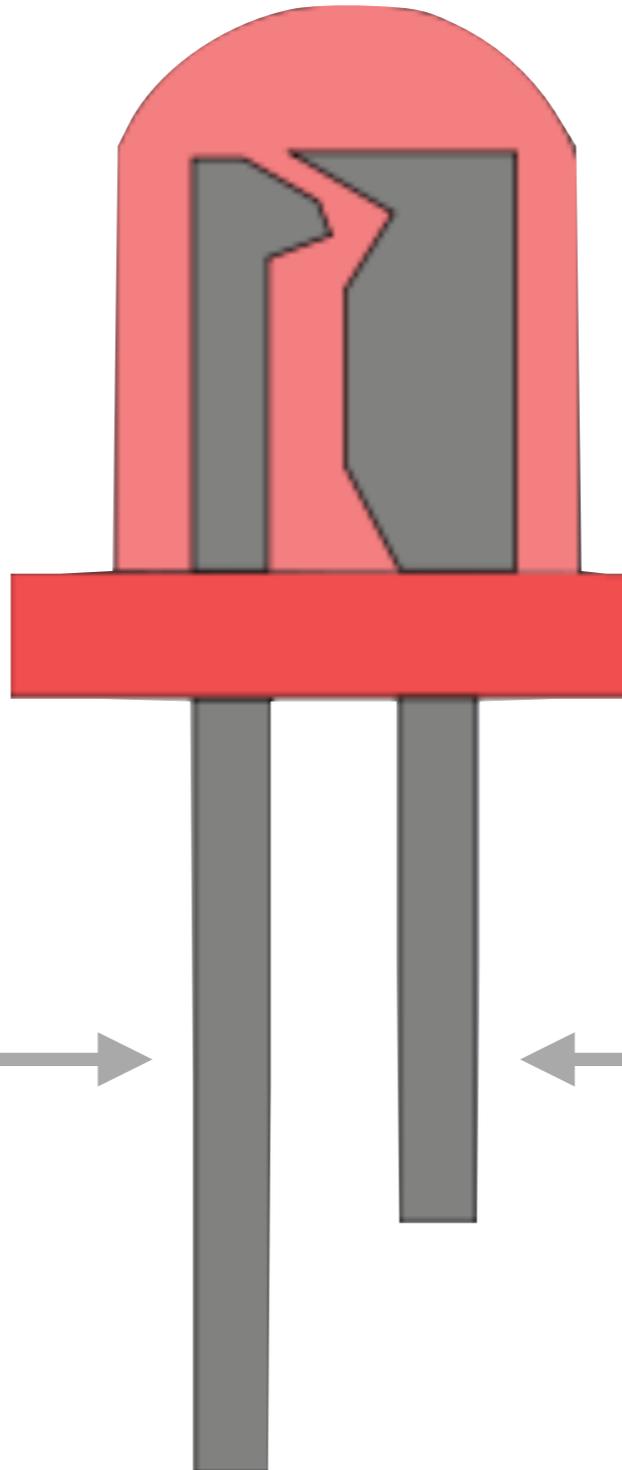


# Beispiel: Ewig Blinken

```
void setup() {  
    // configure PIN 13 (built-in LED) as output  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    // turn the LED on (HIGH is the voltage level)  
    digitalWrite(13, HIGH);  
  
    // wait for a second  
    delay(1000);  
  
    // turn the LED off by making the voltage LOW  
    digitalWrite(13, LOW);  
  
    // wait for a second  
    delay(1000);  
}
```

*Demo*

# Eine Leuchtdiode



**Anode (+)** →

- langes Bein
- runde Seite

← **Kathode (-)**

- kurzes Bein
- abgeflachte Seite

# LED anschließen

- Um eine LED an 5V anzuschließen, braucht man einen *Vorwiderstand* (meist  $\sim 120\Omega$ )
- In unseren LEDs ist dieser Vorwiderstand bereits integriert
- Kathode (–, kurzes Bein) an GND, Anode (+, langes Bein) an Port

*Demo*

# Die Wilde 13

- Wird die LED an einen anderen Port angeschlossen, muss man im gesamten Programm die Portnummer ändern
- In einem großen Programm wird das schnell zum Problem
- Lösung: *Variablen*

# Variablen

- *Variablen* dienen dazu, *Werte* zu speichern.
- Mit der Anweisung

```
int led = 13;
```

wird `led` als eine *Variable* eingeführt, die mit dem Wert 13 belegt ist.

- Nach der Anweisung steht `led` stellvertretend für den Variablenwert

# Typen

- Der *Typ* einer Variable bestimmt, welche Werte die Variable speichern kann
- `int` – ganzzahlige Werte (integer)
- Weitere Typen: `float`, `char`, `void`

# Symbolisches Blinken

```
// Pin 13 has an LED connected on most  
// Arduino boards. Give it a name:
```

```
int led = 13;
```

```
void setup() {  
    pinMode(led, OUTPUT);  
}
```

```
void loop() {  
    digitalWrite(led, HIGH);  
    delay(1000);  
    digitalWrite(led, LOW);  
    delay(1000);  
}
```

# Schneller Blinken

```
// Pin 13 has an LED connected on most  
// Arduino boards. Give it a name:
```

```
int led = 13;
```

```
// Blinking delay (in ms)
```

```
int blink_delay = 250;
```

```
void setup() {  
    pinMode(led, OUTPUT);  
}
```

```
void loop() {  
    digitalWrite(led, HIGH);  
    delay(blink_delay);  
    digitalWrite(led, LOW);  
    delay(blink_delay);  
}
```

*Demo*

# Wechselblinken

```
int led_red    = 12;
int led_green  = 13;

void setup() {
    pinMode(led_red, OUTPUT);
    pinMode(led_green, OUTPUT);
}

void loop() {
    digitalWrite(led_red, HIGH);
    digitalWrite(led_green, LOW);
    ...
}
```

*Demo*

# Bezeichner

- Alle Namen für Variablen und Funktionen (*Bezeichner*) bestehen aus a–z, A–Z, 0–9 und \_ (Unterstrich)
- Bezeichner dürfen nicht mit 0–9 beginnen
- In einem Sketch darf jeder Bezeichner nur 1x vergeben werden

# Bezeichner

- `de lay`, `De lay` und `DE LAY` sind unterschiedliche Bezeichner
  - Konvention:
    - `De lay` – eine *Klasse*
    - `DE LAY` – ein *Makro*
    - `_de lay` – *intern*
- } machen wir nicht!

# So was dähmliches

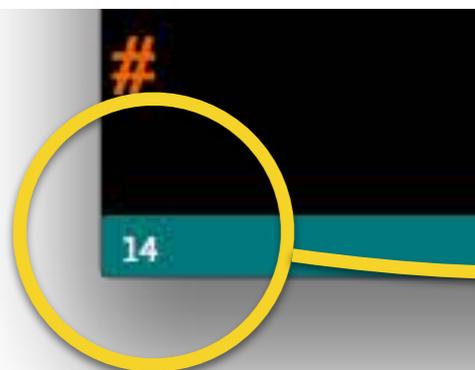
- Bei Fehlern: *Fehlermeldung*

Zeile      Spalte

Blink.ino:7:5: error:  
redefinition of 'int on\_delay'

Fehlermeldung

aktuelle Zeile

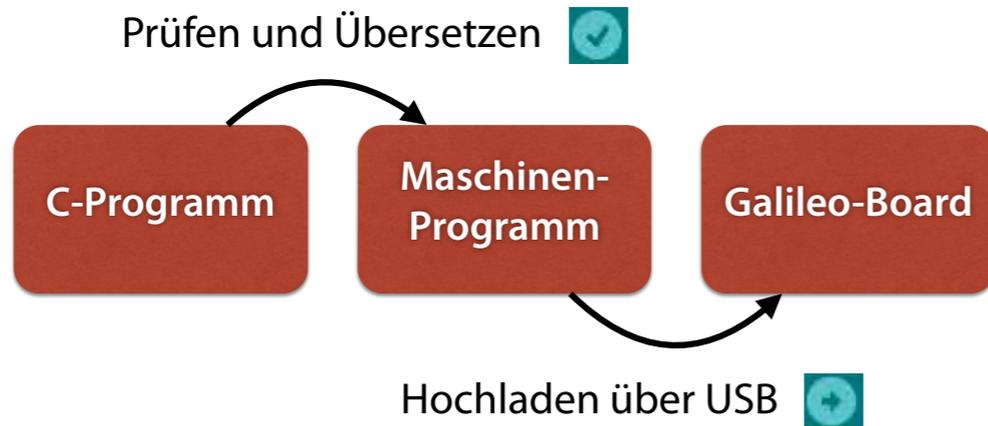


*Demo*

# Vorschau

- Morse-Code
- Funktionen mit Parametern
- Kontrollstrukturen

# Vom Programm zum Prozessor



# Funktionsaufrufe

- Die meisten Funktionen haben *Parameter*, die ihre Funktionsweise bestimmen

`digitalWrite(pin_number, value)`

- Beim Aufruf muss für jeden Parameter ein Wert (*Argument*) angegeben werden

`digitalWrite(13, HIGH);`

Funktionsname → `digitalWrite`  
Wert für *pin\_number* → `13`  
Wert für *value* → `HIGH`

# Variablen

- Variablen* dienen dazu, *Werte* zu speichern.
- Mit der Anweisung

```
int led = 13;
```

wird `led` als eine *Variable* eingeführt, die mit dem Wert 13 belegt ist.

- Nach der Anweisung steht `led` stellvertretend für den Variablenwert

# Symbolisches Blinken

```
// Pin 13 has an LED connected on most  
// Arduino boards. Give it a name:  
int led = 13;
```

```
void setup() {  
    pinMode(led, HIGH);  
}
```

```
void loop() {  
    digitalWrite(led, HIGH);  
    delay(1000);  
    digitalWrite(led, LOW);  
    delay(1000);  
}
```