Hinter den Kulissen

Programmieren für Ingenieure
Sommer 2014

Andreas Zeller, Universität des Saarlandes

LCD einrichten

```
// Anschlüsse der LCD-Anzeige
int rsPin = 2;
int ePin  = 3;
int d4Pin = 4;
int d5Pin = 5;
int d6Pin = 6;
int d7Pin = 7;

LiquidCrystal lcd(rsPin, ePin,
                  d4Pin, d5Pin, d6Pin, d7Pin);

void setup() {
    lcd.begin(16, 2); // Breite, Höhe
    lcd.print("Hello, world!");
}
```

Drehimpulsgeber

- Beim Drehen kurzer Impuls (= Verbindung zwischen *Pin Impuls* und *Impuls links* bzw. *Impuls rechts*)
- Beim Drücken wie Taster

Zeichenketten in C

- Ein einzelnes Zeichen wird in C in einzelne Hochkomma eingeschlossen:

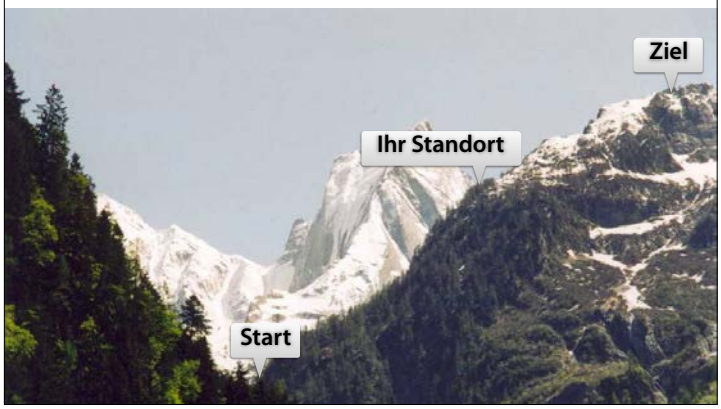
```
char c = 'a';
lcd.print(c);
```

- Die wichtigste Verwendung ist als *Feld* von Zeichen (*Zeichenkette*, auch *String*)
- Zeichenketten enden mit einem speziellen "Null-Zeichen", geschrieben als '\0'

Menü mit Preis

```
int DRINKS = 3;
char *drink_name[] = { "Wasser", "Limo", "Bier" };
double drink_price[] = { 1.00, 1.50, 2.50 };

void print_prices() {
    int x = 0;
    for (int i = 0; i < DRINKS; i++)
    {
        char buffer[100];

        lcd.setCursor(x, 1);
        sprintf(buffer, "%.2f", drink_price[i]);
        lcd.print(buffer);
        x += strlen(drink_name[i]) + 1;
    }
}
```

Bergfest



Lassen Sie uns die Aussicht genießen und etwas zurückblicken :-)

http://commons.wikimedia.org/wiki/File:Piz_Badile,_winter.jpg

Themen heute

- Maschinenmodell
- Programmablauf
- Speicher
- Ultraschall!

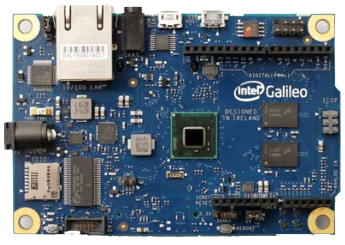
http://commons.wikimedia.org/wiki/File:Media-markt-automat.jpg

Blinken mit Millis

```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Was passiert hier?

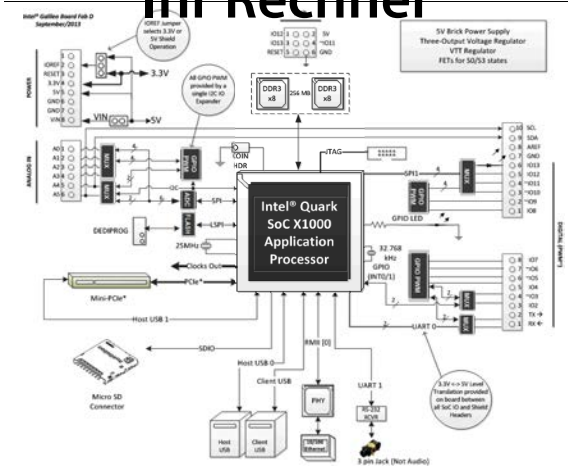
```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```



Wir haben das Programm, und wir haben den Rechner. Wie führt der Rechner das Programm aus?

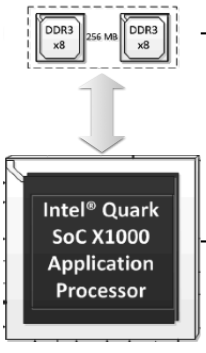
Ihr Rechner

Ihr Rechner



https://communities.intel.com/docs/DOC-21822

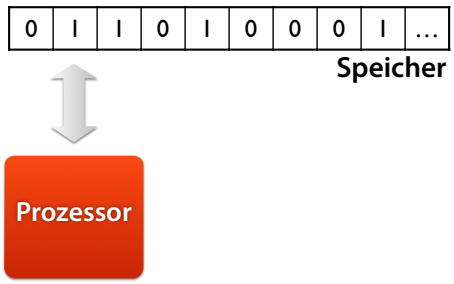
Ihr Rechner



Speicher
enthält *Daten*

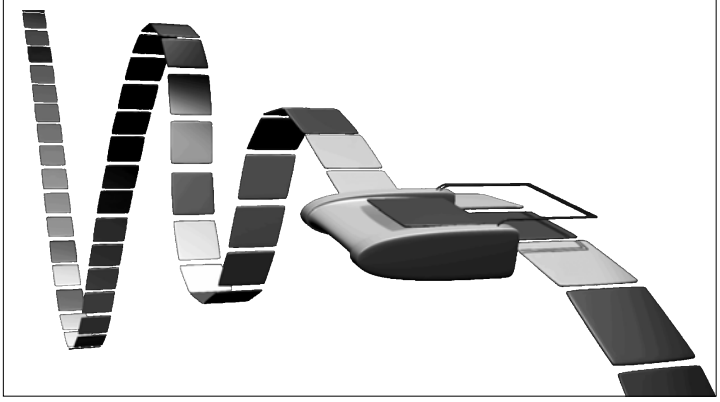
Prozessor
führt *Anweisungen* aus

Ihr Rechner



Wir können noch weiter abstrahieren

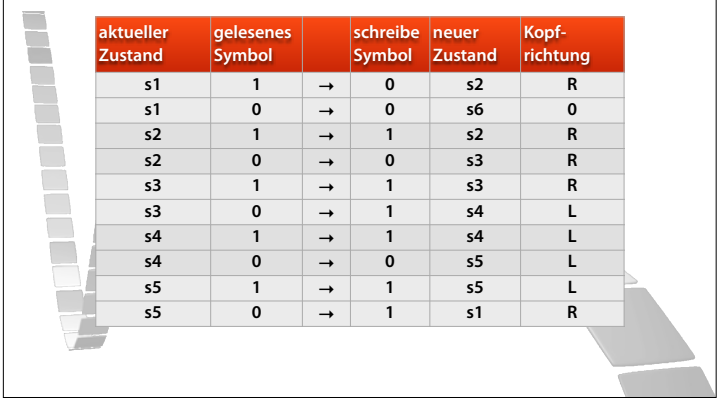
Turingmaschine



Eine Turingmaschine ist ein wichtiges Rechnermodell der Theoretischen Informatik. Sie modelliert die Arbeitsweise eines Computers auf besonders einfache und mathematisch gut zu analysierende Weise. Eine Berechnung besteht dabei aus schrittweisen Manipulationen von Symbolen bzw. Zeichen, die nach bestimmten Regeln auf ein Speicherband geschrieben und auch von dort gelesen werden. Ketten dieser Symbole können verschieden interpretiert werden, unter anderem als Zahlen. Damit

---

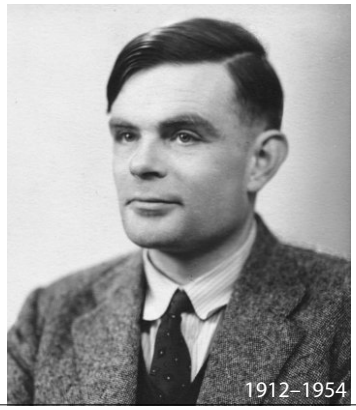
Ein Turing-Programm



aktueller Zustand	gelesenes Symbol		schreibe Symbol	neuer Zustand	Kopf-richtung
s1	1	→	0	s2	R
s1	0	→	0	s6	0
s2	1	→	1	s2	R
s2	0	→	0	s3	R
s3	1	→	1	s3	R
s3	0	→	1	s4	L
s4	1	→	1	s4	L
s4	0	→	0	s5	L
s5	1	→	1	s5	L
s5	0	→	1	s1	R

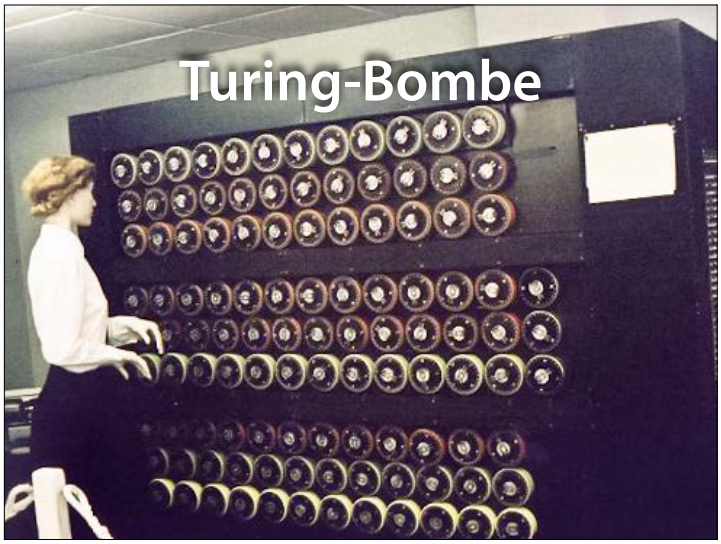
Verdoppelt die Anzahl der Einsen auf dem Band (Quelle: Wikipedia)

Alan Turing



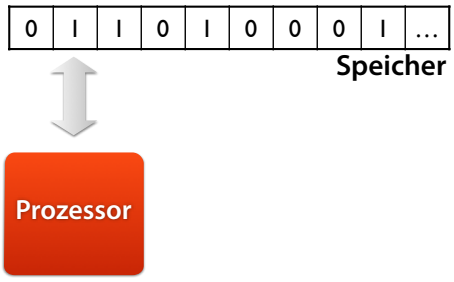
Alan Mathison Turing (* 23. Juni 1912 in London; † 7. Juni 1954 in Wilmslow, Cheshire) war ein britischer Logiker, Mathematiker, Kryptoanalytiker und Informatiker. Er gilt heute als einer der einflussreichsten Theoretiker der frühen Computerentwicklung und Informatik. Turing schuf einen großen Teil der theoretischen Grundlagen für die moderne Informations- und Computertechnologie. Das von ihm entwickelte Berechenbarkeitsmodell der Turingmaschine bildet eines der Fundamente der theoretischen Informatik. Während des Zweiten Weltkrieges war er maßgeblich an der Entzifferung der mit der Enigma verschlüsselten deutschen Funksprüche beteiligt. Nach ihm benannt sind der Turing Award, die bedeutendste Auszeichnung in der Informatik, sowie der Turing-Test zum Nachweis künstlicher Intelligenz. (Wikipedia)

Die ENIGMA (griechisch αἴνιγμα ainigma „Rätsel“) ist eine Rotor-Schlüsselmaschine, die im Zweiten Weltkrieg zur Verschlüsselung des Nachrichtenverkehrs des deutschen Militärs verwendet wurde. Trotz mannigfaltiger Verbesserungen der Verschlüsselungsqualität der Maschine vor und während des Krieges, gelang es den Alliierten mit hohem Aufwand zur Entzifferung, die deutschen Funksprüche nahezu kontinuierlich zu brechen. (Wikipedia)



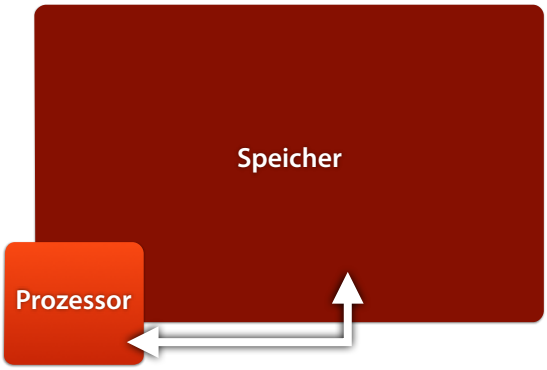
Mit Hilfe der Turing-Bombe (hier ein Nachbau in Bletchley Park, bedient von einer „Wren“) schrumpft der vorher noch so gigantisch erscheinende Schlüsselraum von etwa $2 \cdot 10^{23}$ Möglichkeiten auf vergleichsweise winzige $120 \cdot 17.576 = 2.109.120$ (gut zwei Millionen) Möglichkeiten (etwa 21 bit), eine Zahl, die auch bereits zu Zeiten des Zweiten Weltkriegs mithilfe der damaligen elektromechanischen Technik exhaustiv (erschöpfend) abgearbeitet werden konnte. (Wikipedia)

Ihr Rechner

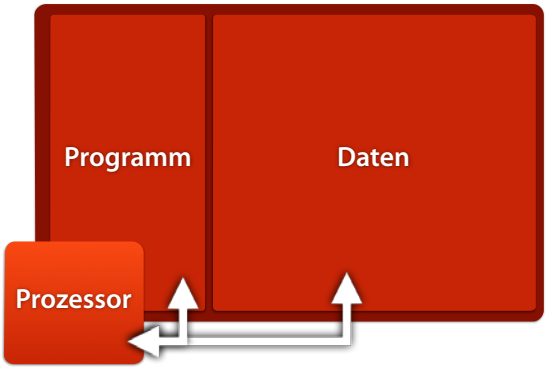


Ihr Rechner

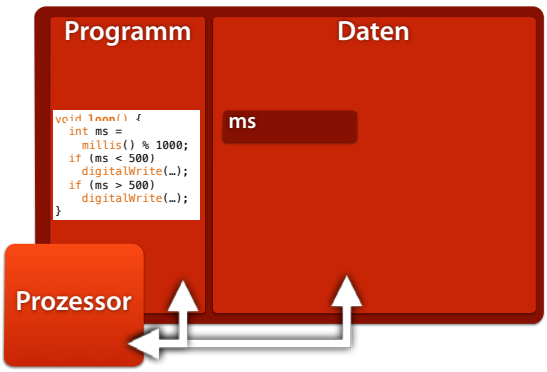
Der Speicher



Der Speicher



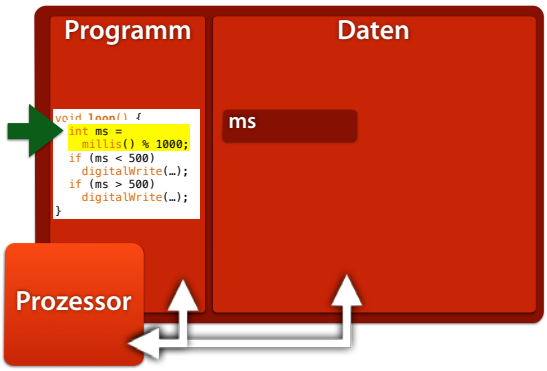
Der Speicher



Der Programmzähler (program counter, PC) zeigt an, welche Anweisung als nächste ausgeführt wird

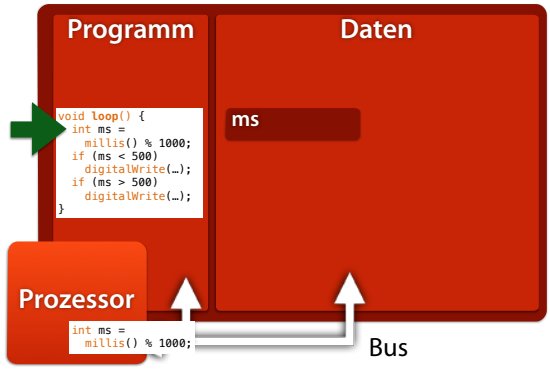
---

Der Programmzähler

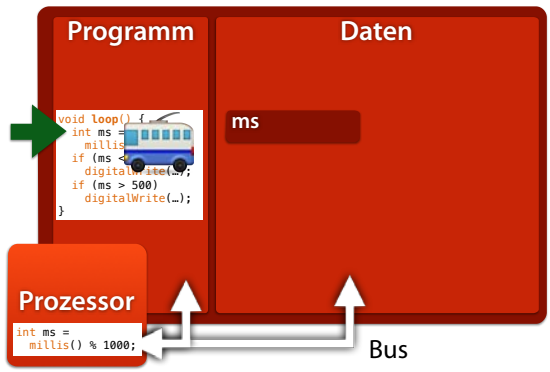


Der Programmzähler (program counter, PC) zeigt an, welche Anweisung als nächste ausgeführt wird

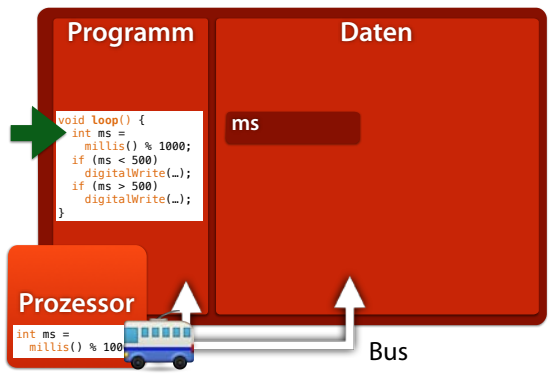
Befehle lesen



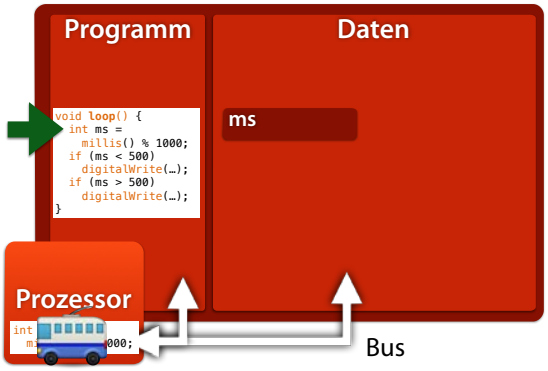
Der Bus



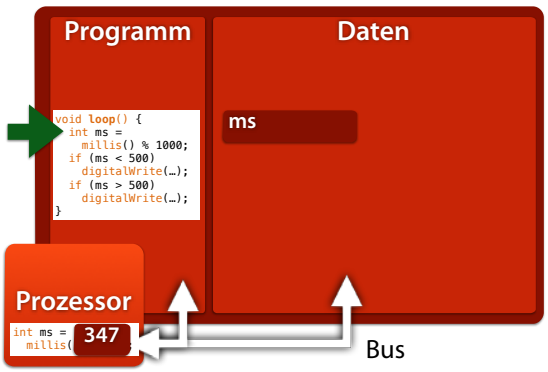
Der Bus



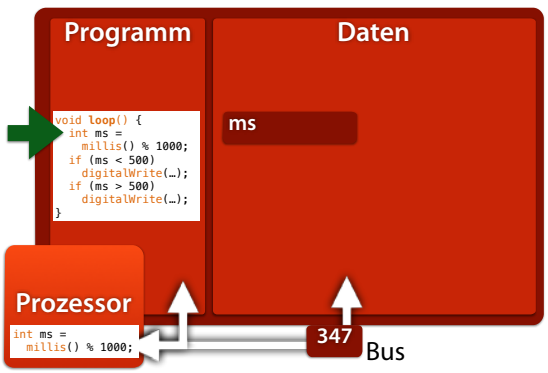
Der Bus



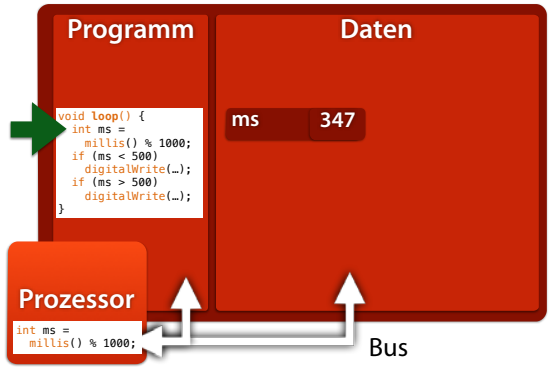
Daten speichern



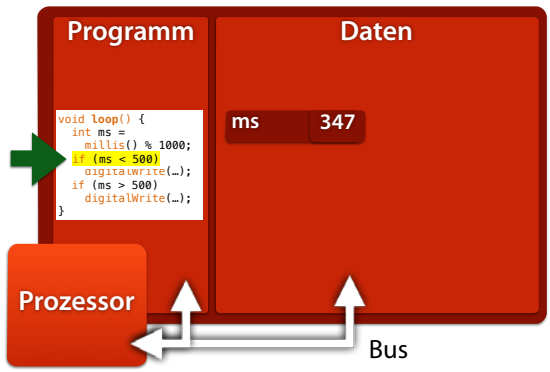
Daten speichern



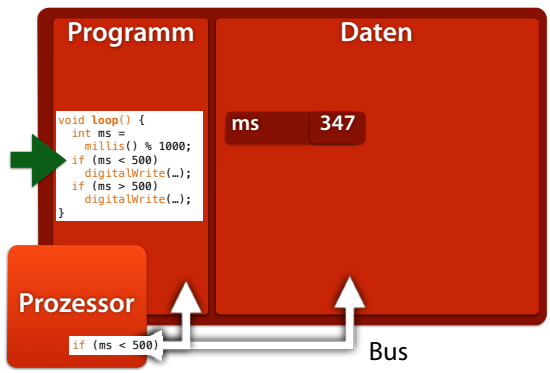
Daten speichern



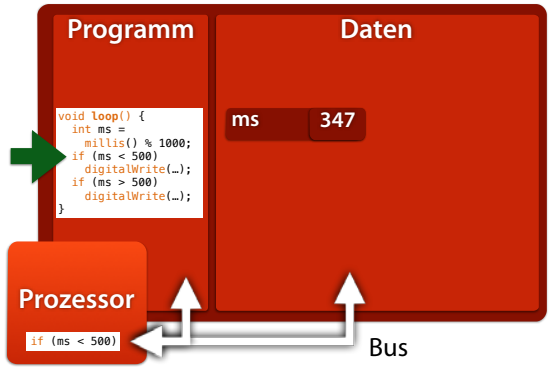
Nächste Anweisung



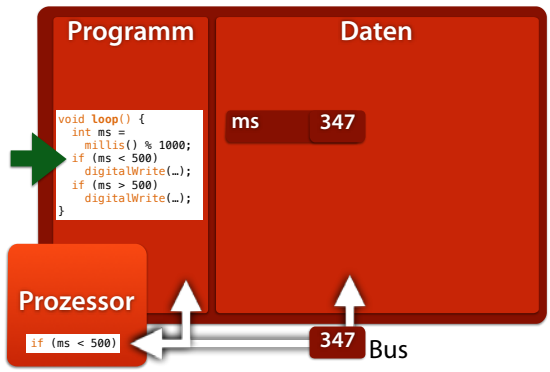
Nächste Anweisung



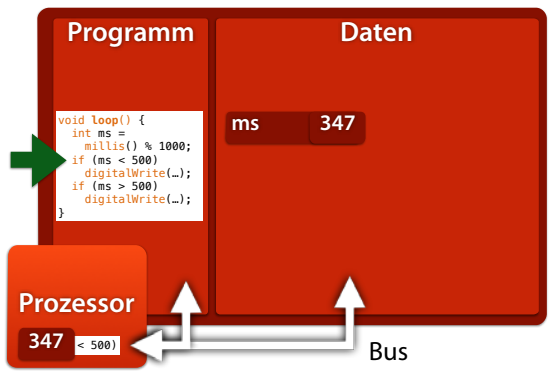
Nächste Anweisung



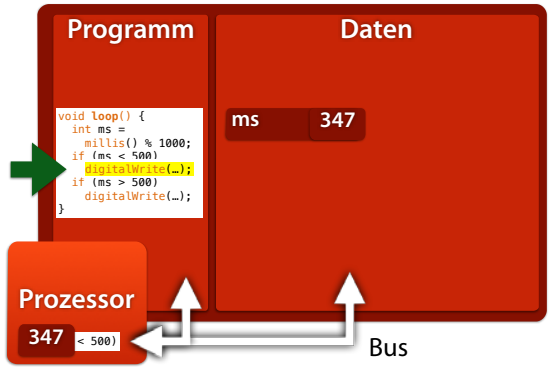
Daten lesen



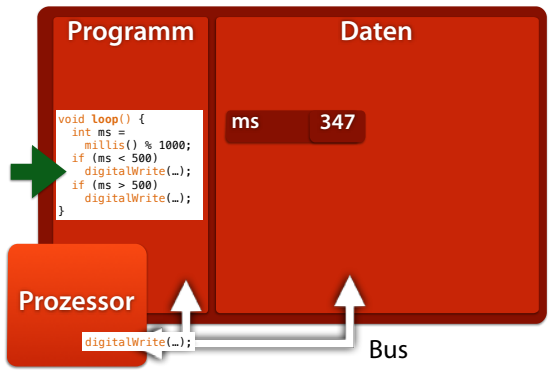
Daten lesen



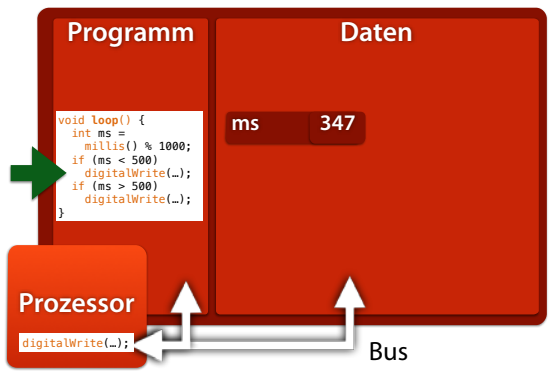
Nächste Anweisung



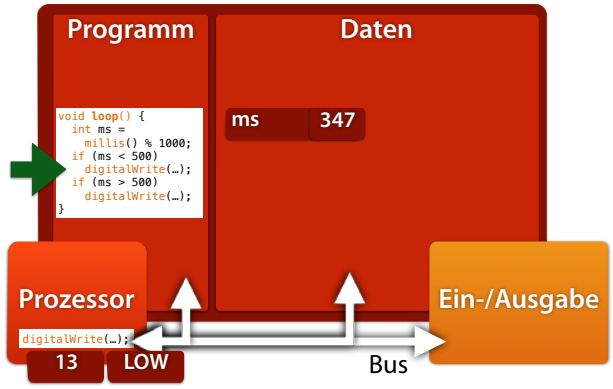
Nächste Anweisung



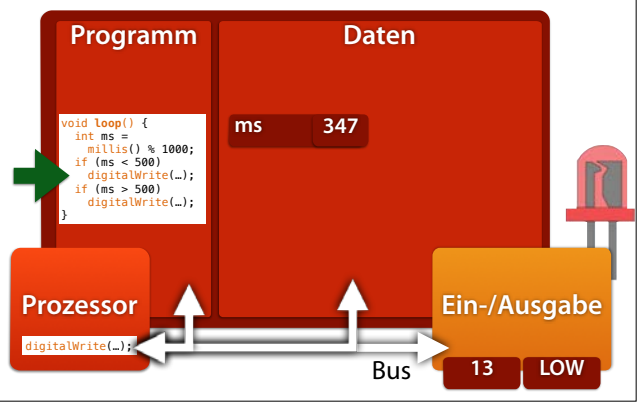
Nächste Anweisung



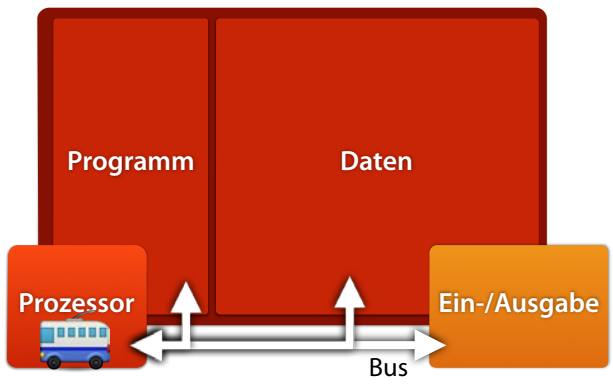
Ein-/Ausgabe



Ein-/Ausgabe



von Neumann-Architektur

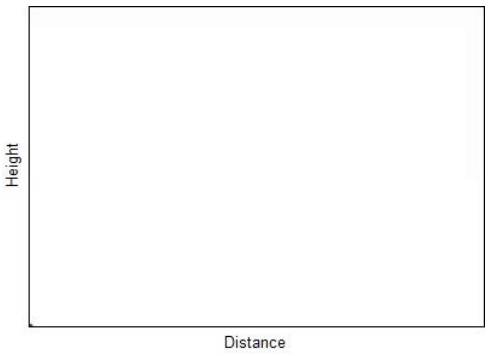


John von Neumann

János Neumann Margittai (* 28. Dezember 1903 in Budapest (Österreich-Ungarn) als János Lajos Neumann; † 8. Februar 1957 in Washington, D.C.) war ein Mathematiker österreichisch-ungarischer Herkunft. Er leistete bedeutende Beiträge zur mathematischen Logik, Funktionalanalysis, Quantenmechanik und Spieltheorie und gilt als einer der Väter der Informatik. Auch an der Weiterentwicklung des amerikanischen Nuklearbomben-Programms bis hin zur

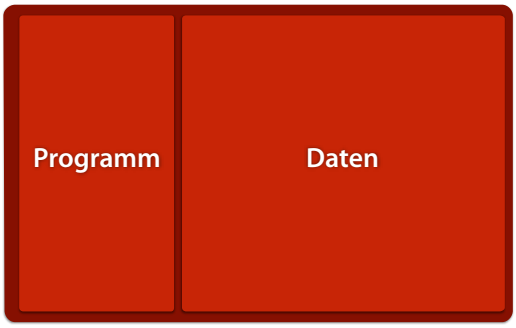
---

Ballistische Tabellen

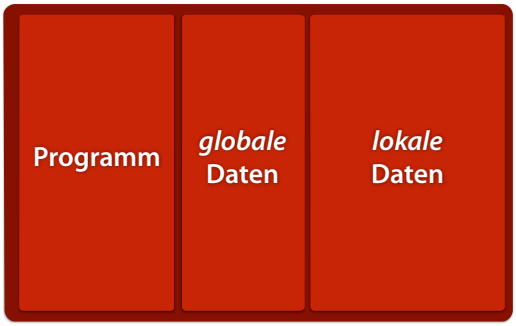


Und was wurde mit diesen ersten Rechnern gemacht? Sie wurden für ballistische Berechnungen genutzt.

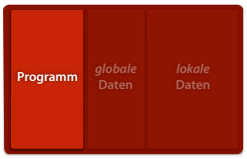
Der Speicher



Der Speicher

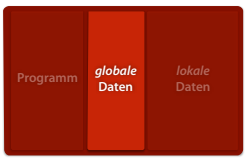


Das Programm



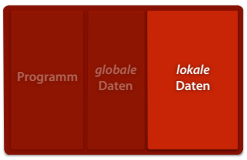
- Liegt wie Daten im Speicher
- Kann nicht auf sich selbst zugreifen oder verändern
- Das *Betriebssystem* lädt und verwaltet Programme im Speicher

Globale Daten



- Enthält Variablen mit *globaler* Sichtbarkeit (= *außerhalb* von Funktionen definiert)
- Von *allen* Funktionen zugänglich

Lokale Daten



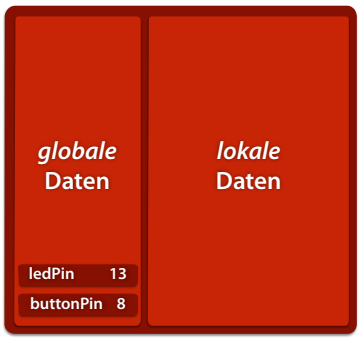
- Enthält Variablen mit *lokaler* Sichtbarkeit (= *innerhalb* von Funktionen definiert)
- Lokale Variablen und Parameter existieren nur *während der Ausführung* der jeweiligen Funktion
- Ein *Funktionsrahmen* speichert diese

Blinken mit Millis

```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Speicherorte

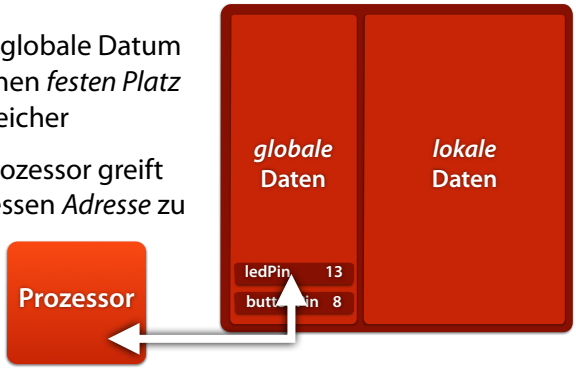
```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```



Zu Beginn haben wir keine lokalen Daten – die werden erst während der Ausführung erzeugt

Speicherorte

- Jedes globale Datum hat einen *festen Platz* im Speicher
- Der Prozessor greift auf dessen *Adresse* zu



Zu Beginn haben wir keine lokalen Daten – die werden erst während der Ausführung erzeugt

---

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

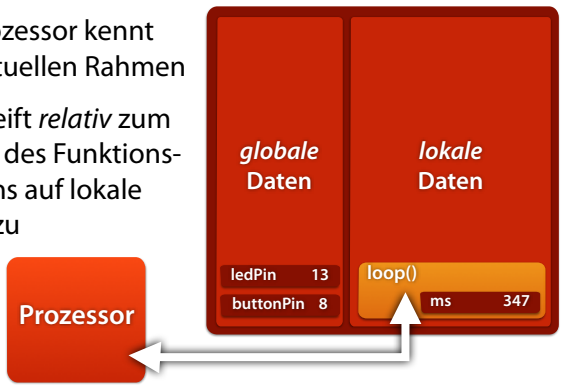
```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Lokale Daten

- Der Prozessor kennt den aktuellen Rahmen
- und greift *relativ* zum Beginn des Funktions-rahmens auf lokale Daten zu



Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

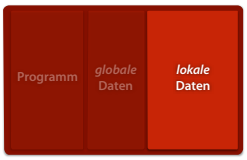
Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsrahmen

```
int ledPin = 13;     // Pin LED
int buttonPin = 8;   // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Funktionsstapel



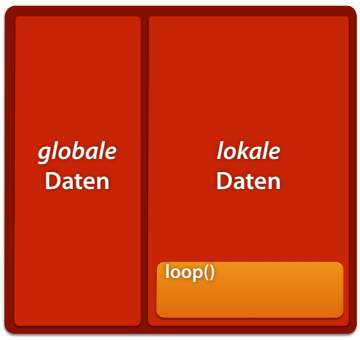
- Die Funktionsrahmen sind als *Stapel* organisiert
- Der oberste Rahmen ist jeweils *aktiv*
- Bei Rückkehr aus der obersten Funktion wird die darunterlegende (aufrufende) Funktion hinter der Aufrufstelle fortgesetzt

Morsecode

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

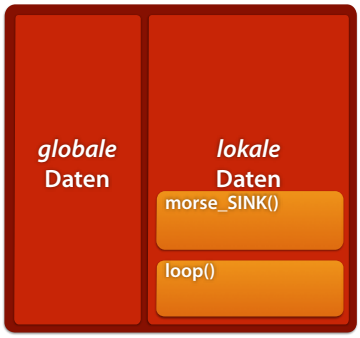


Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

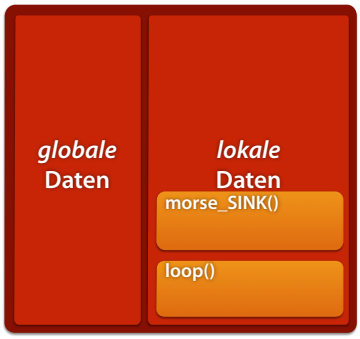
Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```



Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```



Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

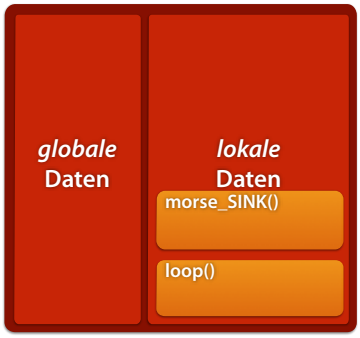
```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

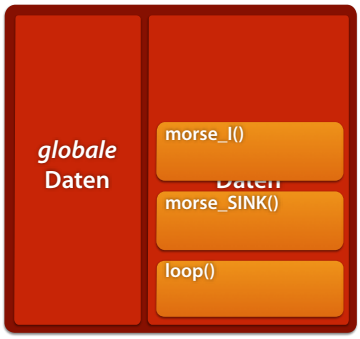


Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```



Funktionsstapel

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

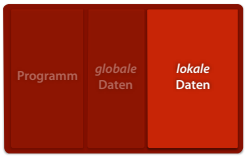
Funktionsstapel

- Aufruf: ablegen *(push)*
- Rückkehr: wegnehmen *(pop)*

Funktionsstapel – analog zu Tablettstapel Mensa

Quelle: http://www.blanco-professional.com/de/catering/produkte/blanco_spender/tablettspender.cfm

Argumente

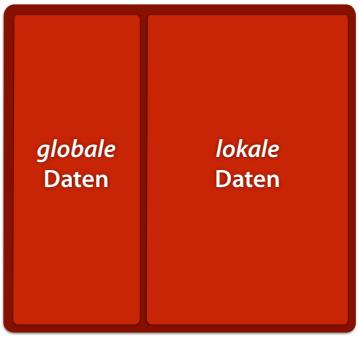


- Beim Aufruf einer Funktion *f* werden *Argumente wie lokale* Variablen abgelegt –
  also im *Funktionsrahmen* von *f*

```
// send n in morse code
void morse_digit(int n) {
  if (n == 0) {
    dah(); dah(); dah(); dah(); dah();
  }
  if (n == 1) {
    dit(); dah(); dah(); dah(); dah();
  }
  // usw. für 2–8
  if (n == 9) {
    dah(); dah(); dah(); dah(); dit();
  }
  pause_letter();
}
```

Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



Zu Beginn haben wir keine lokalen Daten – die werden erst während der Ausführung erzeugt

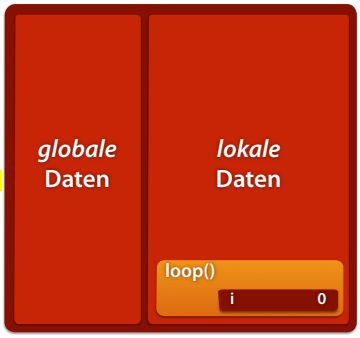
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```


Zu Beginn haben wir keine lokalen Daten – die werden erst während der Ausführung erzeugt

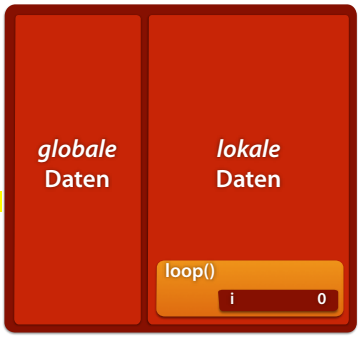
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```

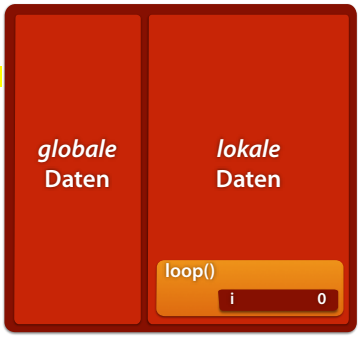


Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```

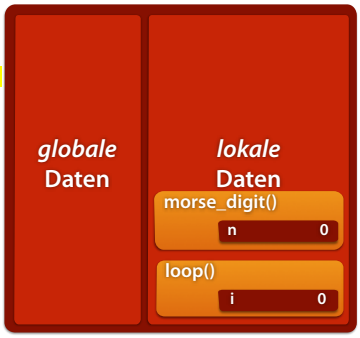
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



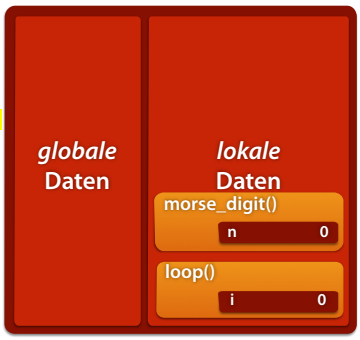
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



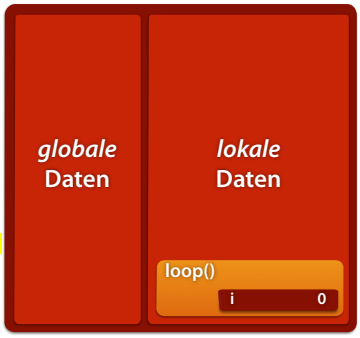
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



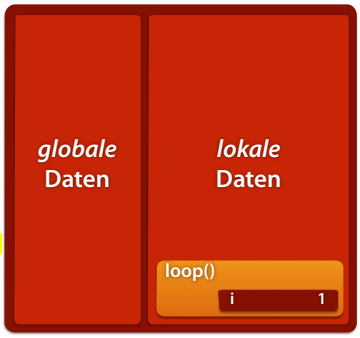
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



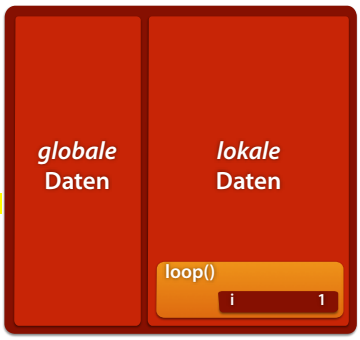
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



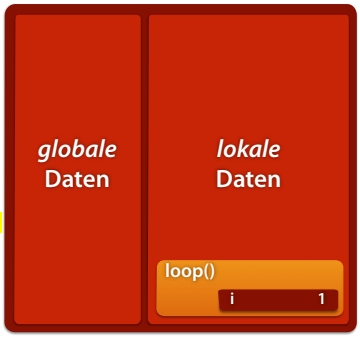
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



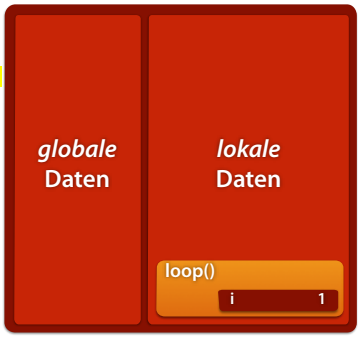
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



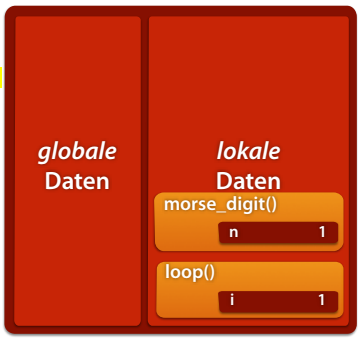
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



Argumente

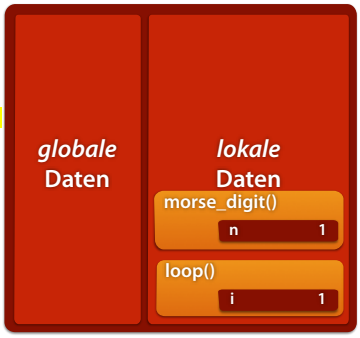
```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



- – – – –

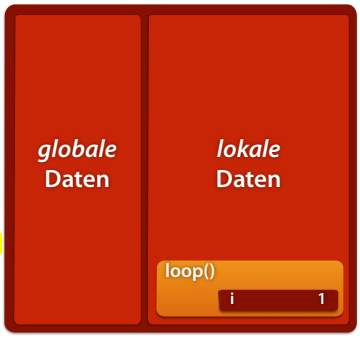
Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



Argumente

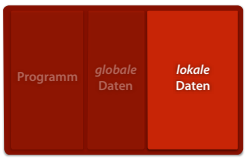
```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```



Argumente

```
void morse_digit(int n) {
  // wie oben
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```

Aktive Funktion



- Das Programm kann nur auf den *jeweils aktiven* (= den obersten) Funktionsrahmen zugreifen
- Ein Aufrufer kann sich darauf verlassen, dass seine lokalen Variablen *unverändert bleiben*

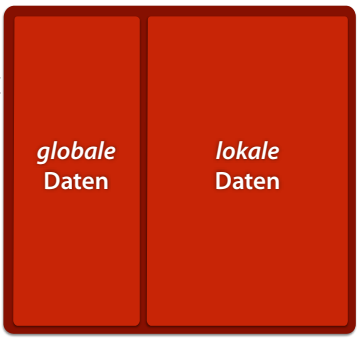
Von Ziffern zu Zahlen

morse_number() gibt eine Zahl *rekursiv* aus:

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```

Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



Zu Beginn haben wir keine lokalen Daten – die werden erst während der Ausführung erzeugt

Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```

Zu Beginn haben wir keine lokalen Daten – die werden erst während der Ausführung erzeugt

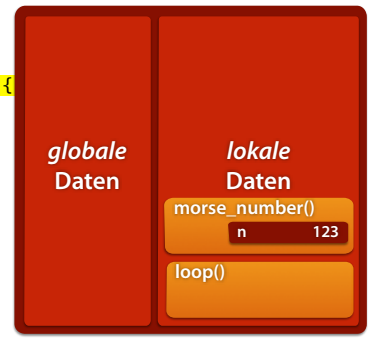
---

Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```

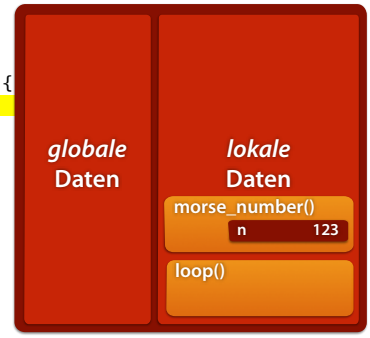
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



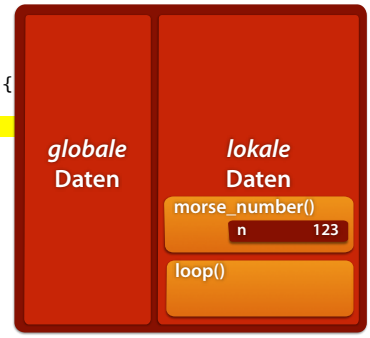
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



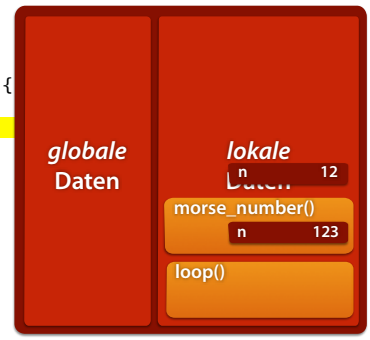
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



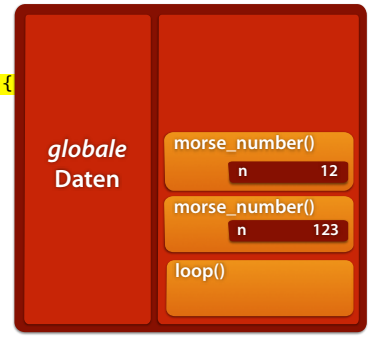
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



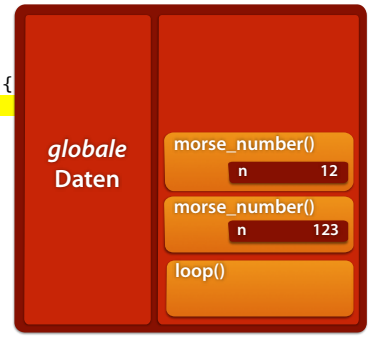
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



Rekursion

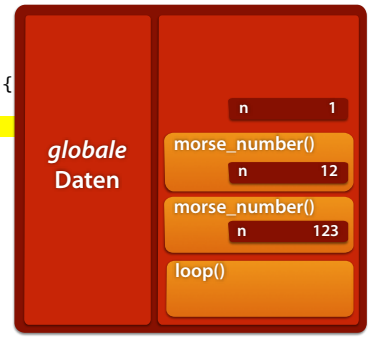
```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



Hier wird auf das **oberste** n der aktiven Funktion zugegriffen – die “unteren”, inaktiven sind nicht zugänglich

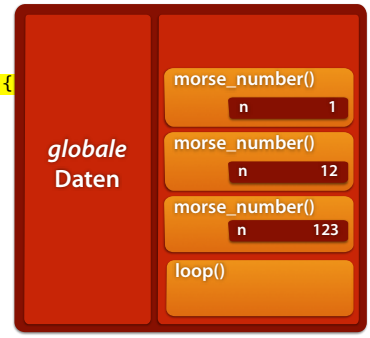
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



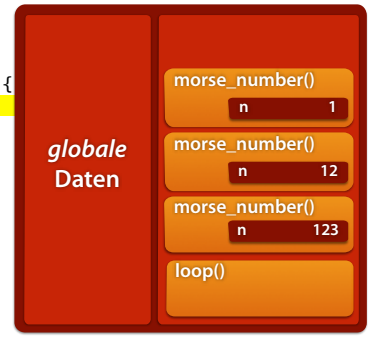
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



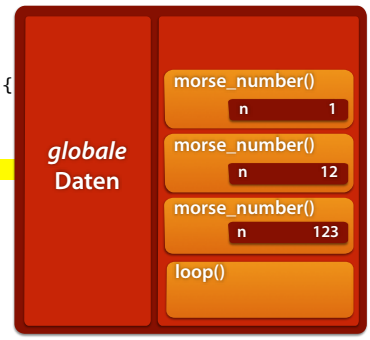
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



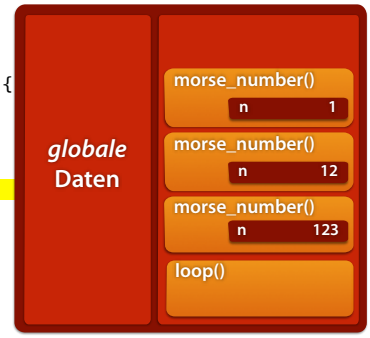
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```

- — — — —

Rekursion

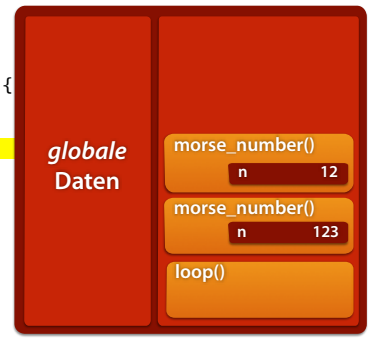
```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



- — — — —

Rekursion

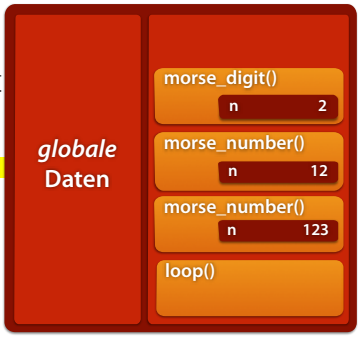
```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



- — — — —
- • — — —

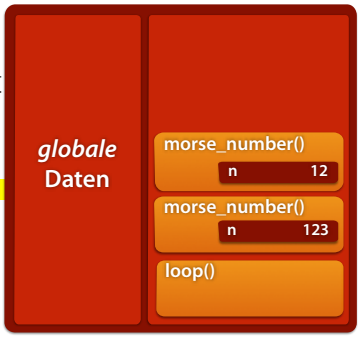
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



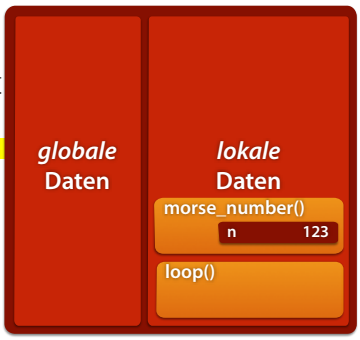
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



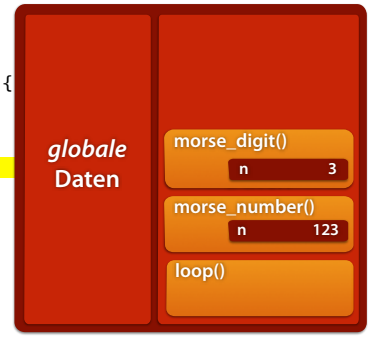
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



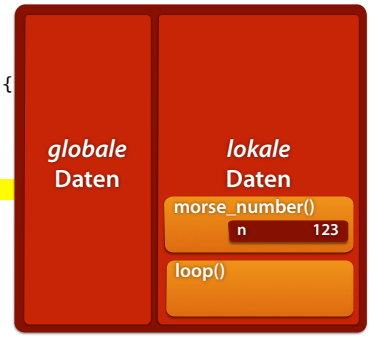
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



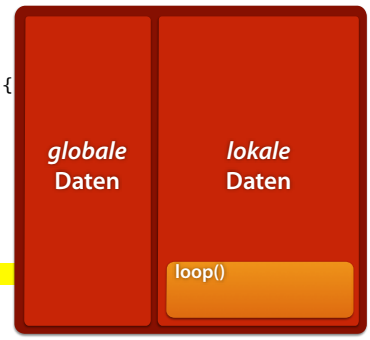
Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```

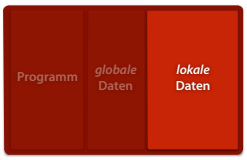


Rekursion

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```



Rückgabewerte



- Gibt eine Funktion einen Wert zurück, legt sie diesen in den lokalen Daten ab
- Die aufrufende Funktion kann sie einer Variablen zuweisen oder in einer Berechnung nutzen

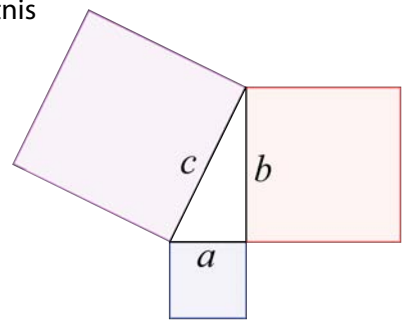
Satz des Pythagoras

Die Seiten eines *rechtwinkligen Dreiecks* stehen im Verhältnis

$$a^2 + b^2 = c^2$$

und somit

$$c = \sqrt{a^2 + b^2}$$



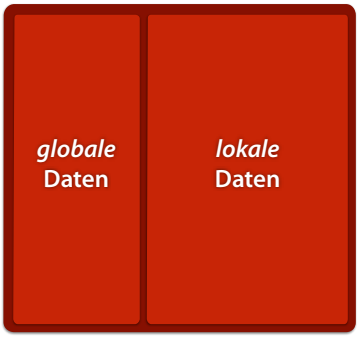
Satz des Pythagoras

pyth(*a*, *b*) berechnet die Seitenlänge *c* eines rechtwinkligen Dreiecks:

```
double pyth(double a, double b) {
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```

Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```

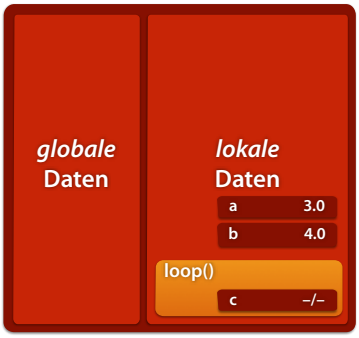
Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```

Zu Beginn hat c noch keinen Wert

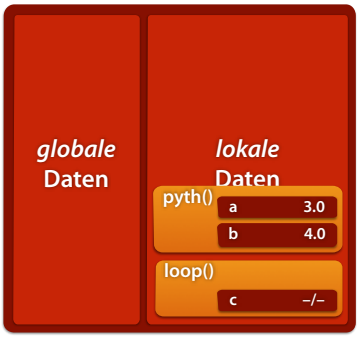
Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



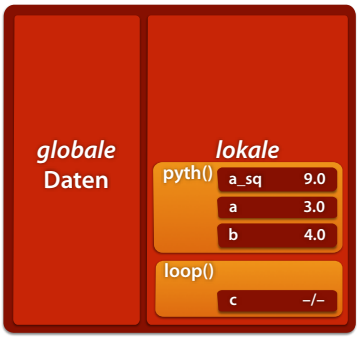
Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



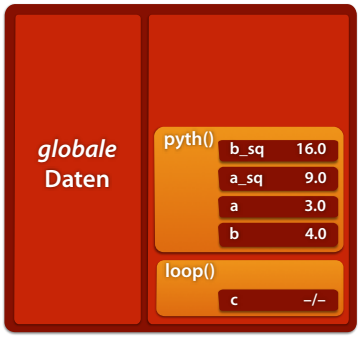
Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



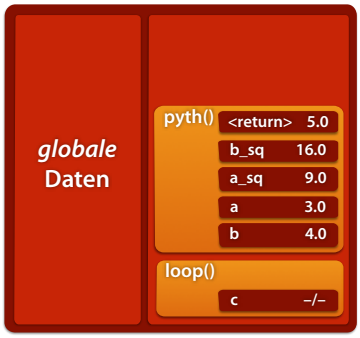
Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



Rückgabewerte

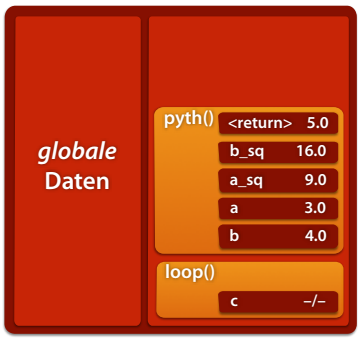
```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



Der Rückgabewert hat keinen eigenen Namen, wird aber lokal gespeichert

Rückgabewerte

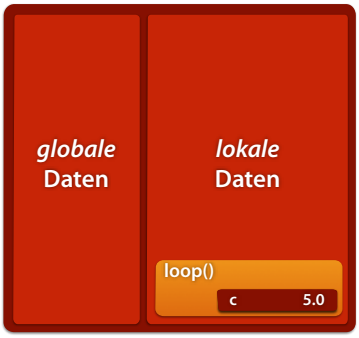
```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



Bei der Rückkehr bleibt der Rückgabewert bis zum Ende der aufrufenden Anweisung erhalten

Rückgabewerte

```
double pyth(double a, double b)
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



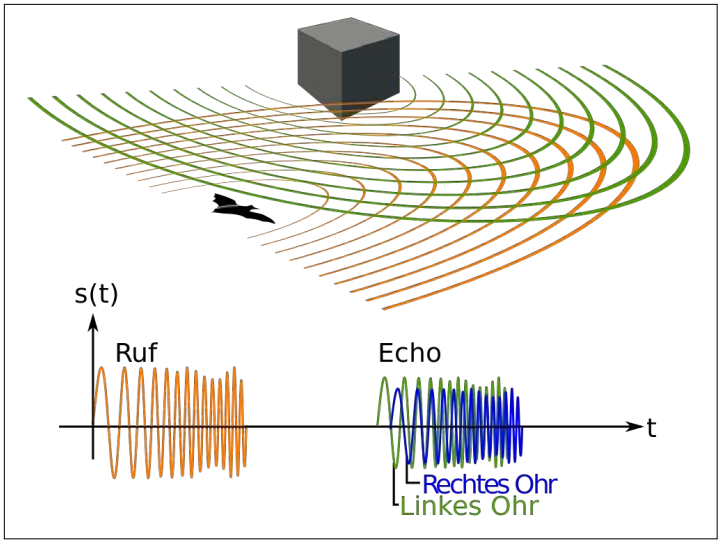
und kann so gespeichert oder genutzt werden

Ausblick

- Zeiger (gemeinsam Speicher nutzen)
- Haldenspeicher (Speicher nach Bedarf)
- Datenstrukturen (für große Datenmengen)
- Konstanten (schreibgeschützte Werte)

Ultraschall!

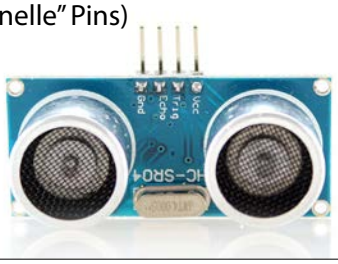
- Der HC-SR04 vereinigt einen Ultraschall-Lautsprecher mit einem Mikrofon
- Kann *Signallaufzeiten* messen



http://commons.wikimedia.org/wiki/File:Animal_echolocation_german.svg

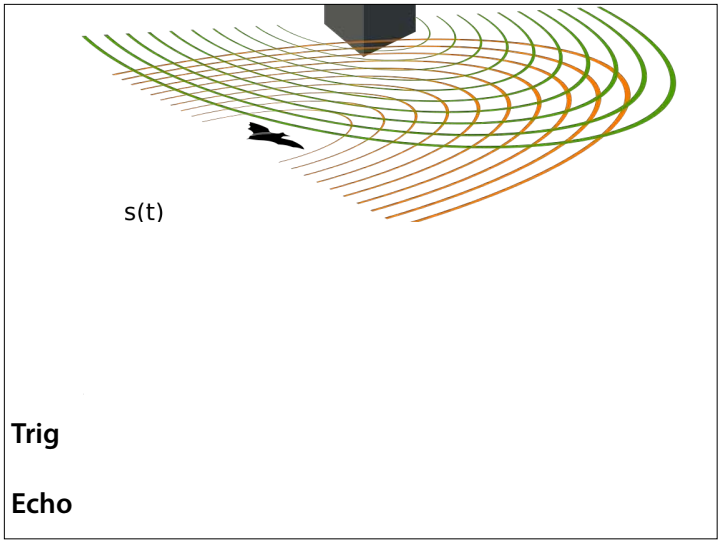
Schaltung

- **Vcc** und **Gnd** werden an + und – angeschlossen
- **Trig** und **Echo** an Pin **2 und 3** des Boards ("schnelle" Pins)



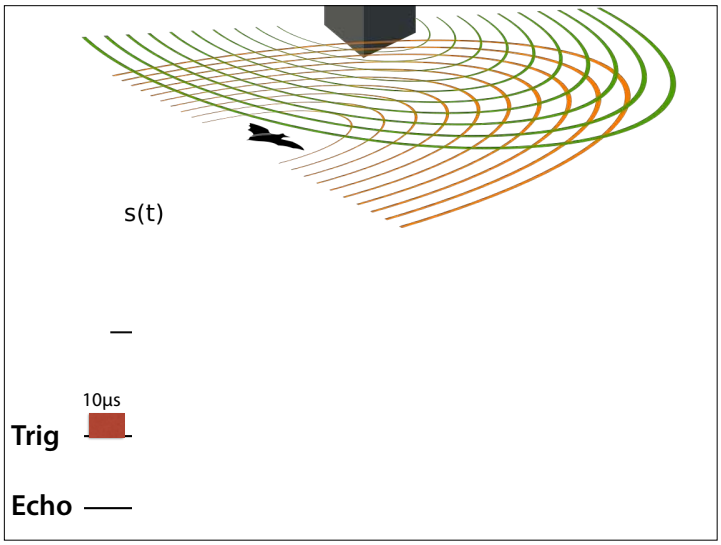
Bedienung

- **Trig** erhält einen 10µs langen HIGH-*Puls*
- Trifft das **Echo** ein, wird es für die Dauer der Signallaufzeit auf HIGH gesetzt



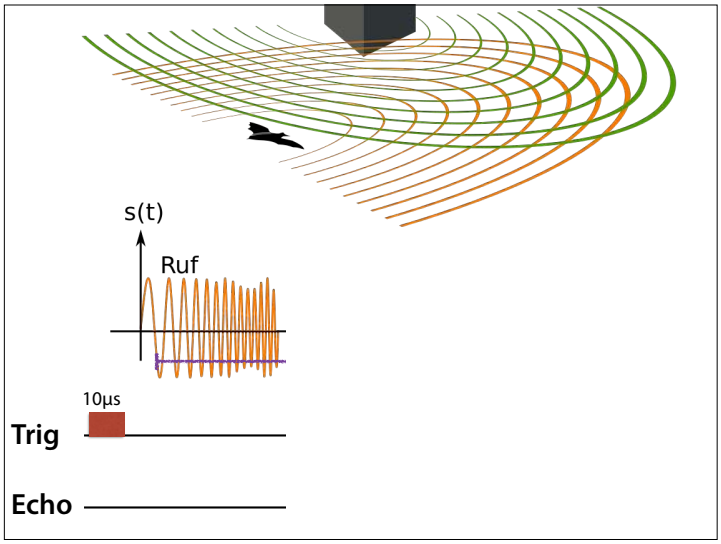
http://commons.wikimedia.org/wiki/File:Animal_echolocation_german.svg

---

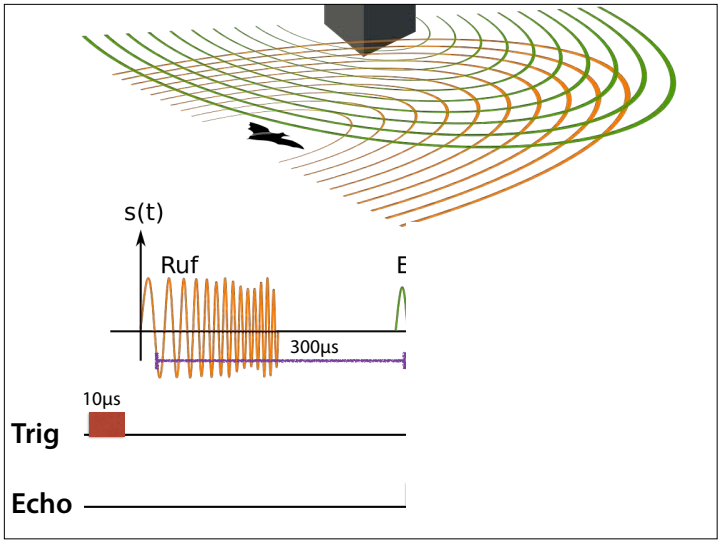


Wir senden zunächst einen Trigger, indem wir **Trig** für 10µs auf **HIGH** setzen

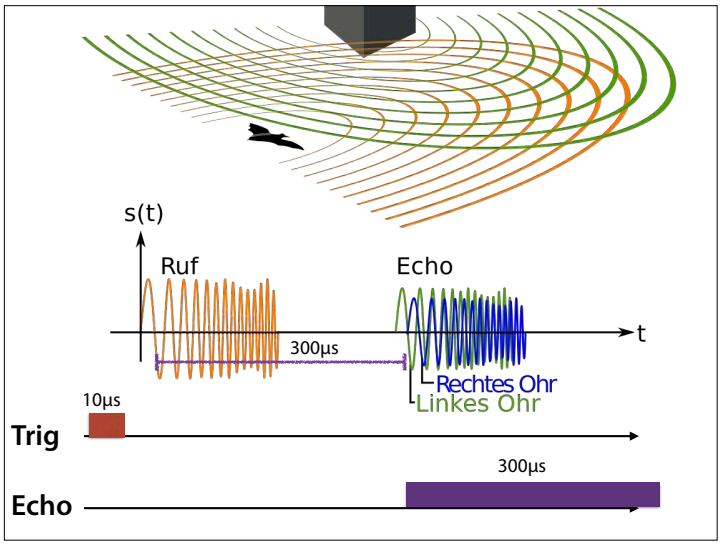
---



Daraufhin sendet der Lautsprecher ein Signal



Je nach Reflexion kommt das Signal zurück (z.B. nach 300 µs)



Dann wird das **Echo**-Signal für die Dauer der Signallaufzeit (also hier auch 300µs) auf **HIGH** gesetzt.

Setup

- **Echo** muss an Pin 2 oder 3 (schnelle I/O), um INPUT_FAST zu nutzen

```
int trigPin = 2;    // Pin Trigger
int echoPin = 3;    // Pin Echo

void setup() {
    Serial.begin(9600);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT_FAST);
}
```

Trigger

- Wir senden ein 10µs langes HIGH-Signal

```
void sendTrigger() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
}
```

Puls

- Wir warten auf einen Übergang von LOW nach HIGH und zurück und messen die Zeit

```
long fetchPulse() {
  while (digitalRead(echoPin) == LOW)
  {}

  unsigned long start = micros();
  while (digitalRead(echoPin) == HIGH)
  {}

  unsigned long end = micros();
  return end - start;
}
```

Ausgabe

- Wir geben die gemessene Zeit aus

```
void loop() {
  sendTrigger();
  long duration = fetchPulse();

  Serial.print(duration);
  Serial.println(" us");
  delay(100);
}
```

Timeout

- Was passiert, wenn der Übergang ausbleibt?
- Wir müssen eine *Endlosschleife* verhindern
- Nach 30ms soll Schluss sein

Puls mit Timeout

```
long fetchPulse() {
  unsigned long wait = micros();
  while (digitalRead(echoPin) == LOW) {
    if (micros() - wait > 30000)
      return -1; // Timeout
  }

  unsigned long start = micros();
  while (digitalRead(echoPin) == HIGH) {
    if (micros() - start > 30000)
      return -1; // Timeout
  }

  unsigned long end = micros();
  return end - start;
}
```

Ausgabe

- Jetzt mit Behandlung von Timeouts

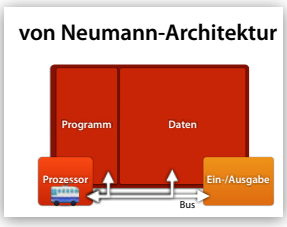
```
void loop() {
  sendTrigger();
  long duration = fetchPulse();
  if (duration < 0)
    return;

  Serial.print(duration);
  Serial.println(" us");
  delay(100);
}
```

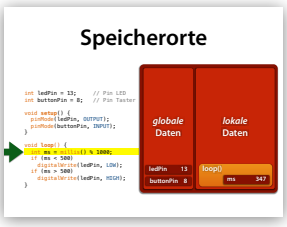
Demo

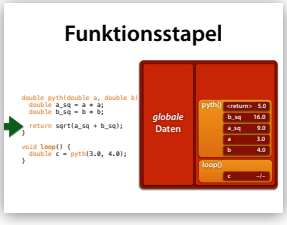
- Schreibmarke bewegen; mit cursor() anzeigen lassen

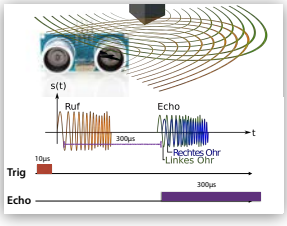
Nächste Schritte

- Umrechnen in *Entfernung* (Schallgeschwindigkeit; kalibrieren)
- Anzeige auf LCD-Display (als Entfernung und/oder Balken)









Handouts

Blinken mit Millis

```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin Taster

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int ms = millis() % 1000;
  if (ms < 500)
    digitalWrite(ledPin, LOW);
  if (ms > 500)
    digitalWrite(ledPin, HIGH);
}
```

Morsecode

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S();
    morse_I();
    morse_N();
    morse_K();
    pause_word();
}

void loop() {
    morse_SINK();
}
```

Ziffern morsen

```
// send n in morse code
void morse_digit(int n) {
  if (n == 0) {
    dah(); dah(); dah(); dah(); dah();
  }
  if (n == 1) {
    dit(); dah(); dah(); dah(); dah();
  }
  // usw. für 2–8
  if (n == 9) {
    dah(); dah(); dah(); dah(); dit();
  }
  pause_letter();
}

void loop() {
  int i = 0;
  while (i < 10) {
    morse_digit(i);
    i = i + 1;
  }
}
```

Von Ziffern zu Zahlen

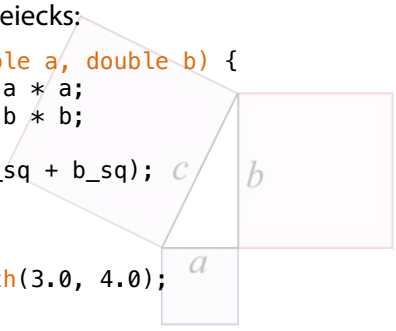
morse_number() gibt eine Zahl *rekursiv* aus:

```
void morse_number(int n) {
  if (n >= 10) {
    morse_number(n / 10);
  }
  morse_digit(n % 10);
}

void loop() {
  morse_number(123);
}
```

Satz des Pythagoras

pyth(*a*, *b*) berechnet die Seitenlänge *c* eines rechtwinkligen Dreiecks:

```
double pyth(double a, double b) {
  double a_sq = a * a;
  double b_sq = b * b;

  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pyth(3.0, 4.0);
}
```



Setup

- **Echo** muss an Pin 2 oder 3 (schnelle I/O), um INPUT_FAST zu nutzen

```
int trigPin = 2;    // Pin Trigger
int echoPin = 3;    // Pin Echo

void setup() {
    Serial.begin(9600);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT_FAST);
}
```

Trigger

- Wir senden ein 10µs langes HIGH-Signal

```
void sendTrigger() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
}
```

Puls mit Timeout

```
long fetchPulse() {
  unsigned long wait = micros();
  while (digitalRead(echoPin) == LOW) {
    if (micros() - wait > 30000)
      return -1; // Timeout
  }

  unsigned long start = micros();
  while (digitalRead(echoPin) == HIGH) {
    if (micros() - start > 30000)
      return -1; // Timeout
  }

  unsigned long end = micros();
  return end - start;
}
```