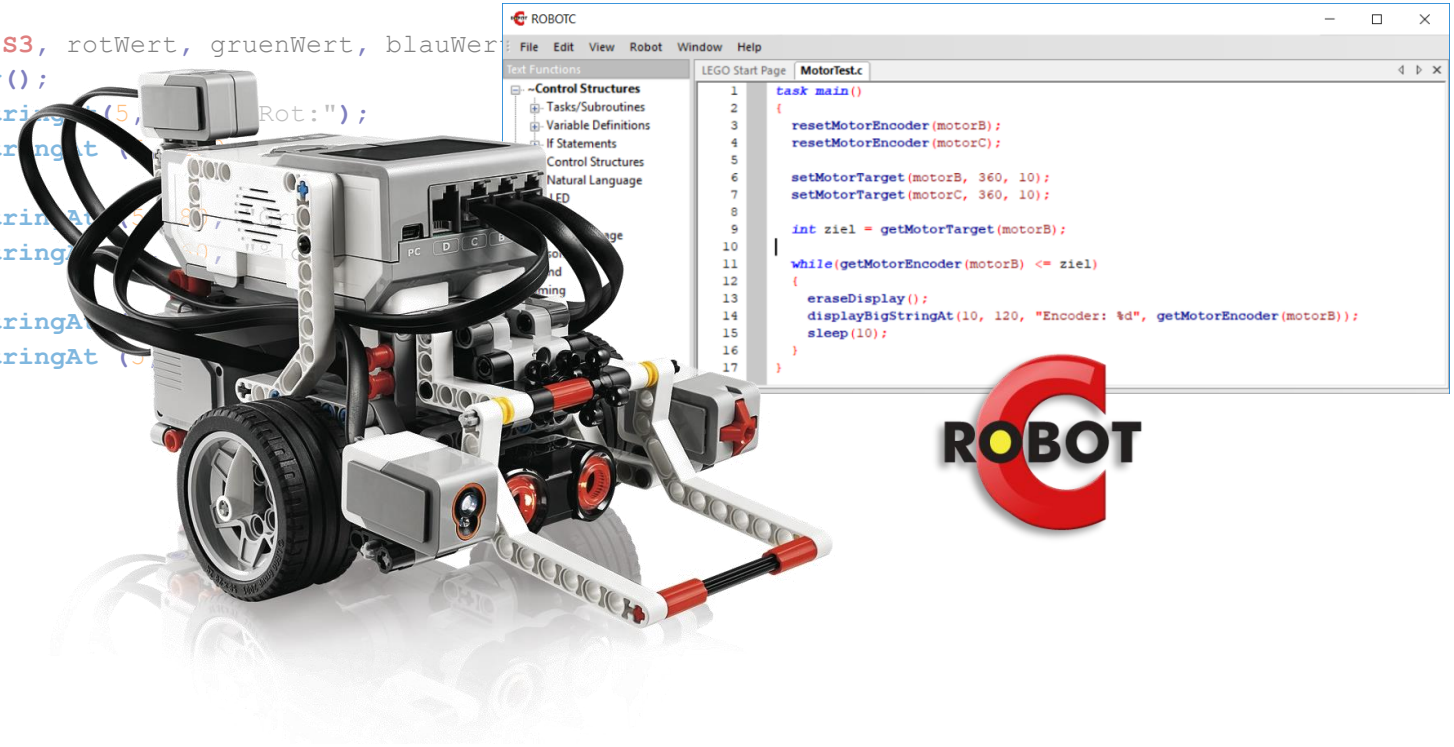
RobotC Einführung für den EV3

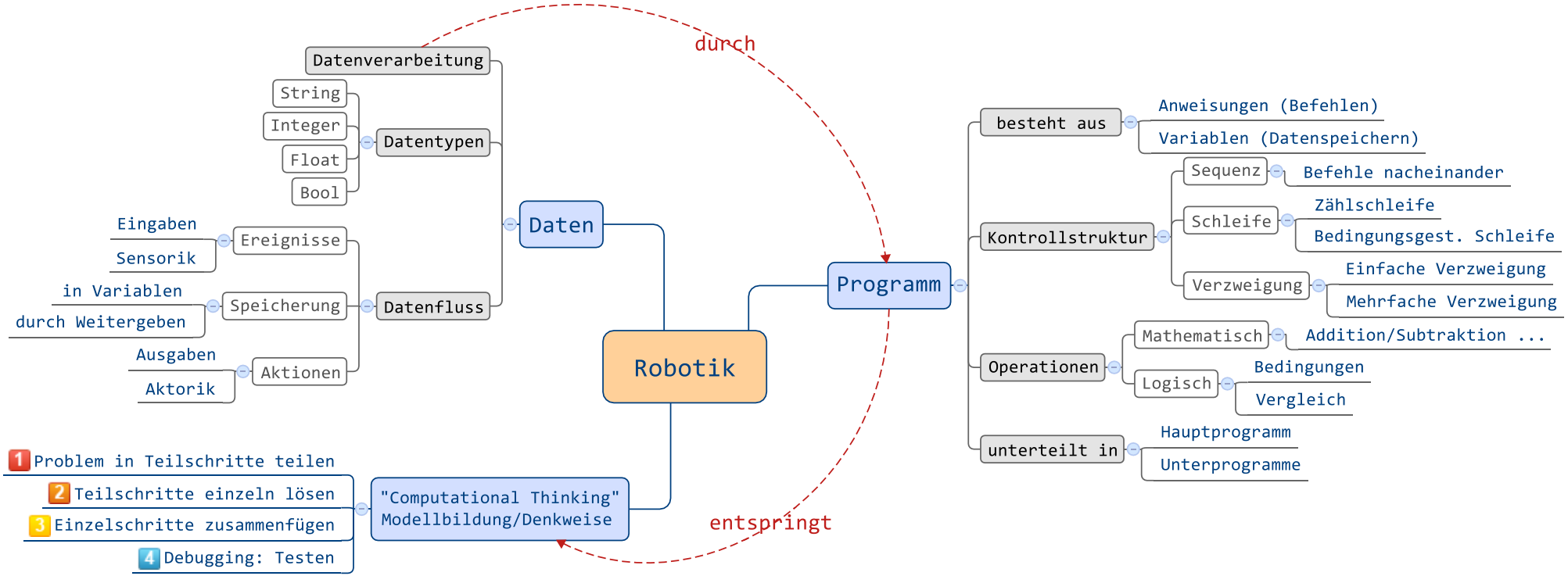
Kepler Gymnasium Weil der Stadt | Thomas Jörg | Stand: 02. September 2018 | Version 1.0

```
task main(){
    SensorType[S3] = sensorEV3_Color;
    SensorType (S3);
    SensorType [S3] = modeEV3Color_Color;

    long rotWert;
    long gruenWert;
    long blauWert;

    while (true)
    {
        getColorRGB(S3, rotWert, gruenWert, blauWer
        eraseDisplay();
        displayBigStri    (5,    Rot:");
        displayBigStr     t (
        displayBigStrin   A
        displayBigString
        displayBigStringA
        displayBigStringAt (
        sleep (50);
    }
}
```



Einleitung

Unabhängig von der Programmiersprache schreibt man ein Programm nach allgemeinen Konzepten, die für alle Programmiersprachen gleich sind:



Ein Roboter (allgemein: Computer) funktioniert nach dem EVA-Prinzip: Daten-Eingabe, Verarbeitung und Ausgabe. Die Verarbeitung erfolgt durch Programme. Diese Programme sind nach allgemein gültigen Prinzipien aufgebaut: Man will ein Problem lösen (z.B. den Weg aus einem Labyrinth finden). Das Gesamtproblem wird in Teilschritte aufgeteilt (z.B. Weg merken, Links/Rechts-Abbiegen, Wegerkennung usw.) Die Lösungen werden programmiert: Für die Teilschritte und das Gesamtproblem.

Die verwendete Programmiersprache ist dabei zweitrangig, sie muss nur genügend Anweisungen verstehen und umsetzen können, um das Problem zu lösen!

Datentypen und Variablen: Wie Computer Daten speichern und verarbeiten

Grundsätzlich speichern (und verarbeiten) Computer „unter der Haube“ ihre Daten **immer** in binärer Form, also indem ein Bit (die kleinste Dateneinheit) die zwei Zustände „0“ und „1“ annehmen kann. Daraus werden komplexe Datentypen zusammengesetzt, die unseren menschlichen Gewohnheiten entsprechen.

Damit sich der Computer diese Daten merken kann, gibt man den Daten einen Namen. Das macht man …

- … so wie man zum Beispiel einer Dose eine Beschriftung gibt, damit man weiss, welche Inhalte darin zu finden sind und man sie wiedererkennt.

- … so wie man zum Beispiel in der Mathematik einem Buchstaben in einer Gleichung einen Zahlenwert zuordnet (zum Beispiel `x = 3` oder `x = 4y - 2` oder ähnliches). Die Schreibweise in RobotC ist der mathematischen Schreibweise „abgeschaut“ und deshalb sehr ähnlich.

Wozu benötigt man Variablen?

In Variablen können zum Beispiel Nutzereingaben, Sensorwerte, Uhrzeiten, erreichte Punktzahlen (bei Computerspielen), Kontostände usw. gespeichert werden.

Umgang mit Variablen

Dieses Namen-Wert-Paar nennt man mit dem Fachbegriff eine „**Variable**“. Mit dem **Namen** der Variablen kann das Programm jederzeit auf den **Wert** der Variablen zugreifen. In der Programmierung bedeutet das, dass eine Variable:

- durch die Vergabe eines Namens erzeugt wird *(Deklarierung),*
- durch Ansprechen des Namens einen ersten Wert zugewiesen bekommt *(Initialisierung)*,
- durch Ansprechen des Namens der Wert ausgelesen werden kann *(Lesen)*,
- dieser Wert jederzeit geändert werden kann *(Zuweisung)*.

Prinzip 1: y = 23	Prinzip 2: y = y + 1
Die Variable mit Namen „y“ bekommt den Wert auf der rechten Seite zugewiesen hier die Ganzzahl „23“, ***wie in der Mathematik üblich.***	Die Variable mit Namen „y“ bekommt den Wert der rechten Seite zugewiesen: ihren bisherigen Wert PLUS 1, ***und zwar anders als in der Mathematik üblich***.

Die vier grundlegenden Datentypen

1: int („Integer-Variable“)

Ganze Zahlen, negativ und positiv

```
int Ganzzahl = 15
```

2: float („Float-Variable“)

Fließkommazahlen, negativ und positiv

```
float Kommazahl = 3.1415926
```

3: string („String-Variable“)

Buchstaben und Text

```
string meinTxt = "Hallo"
```

4: boolean („Boolsche Variable“)

Zwei logische Werte „True“ & „False“

```
bool JaOderNein = True
```

Bei RobotC gilt: Einer Variablen wird bei der Initialisierung ein bestimmter Datentyp zugeordnen. Dieser einmal definierte Datentyp kann nicht mehr verändert werden. Man nennt das „strikte Typisierung“. Die folgende Sequenz z.B. ergibt einen Fehler:

```
int meineVariable = 15
meineVariable = „ein neuer Text“
```

Wichtig zu wissen: Verarbeitung im Prozessor

Die CPU (also der Prozessor, das „Gehirn“ des Computers) besitzt eigene, baulich voneinander abgetrennte Teile zur Verarbeitung von Ganz- und Kommazahlen. Ganzzahlen werden schnell und genau, Kommazahlen eher langsam und unpräzise verarbeitet.

Aufgabe zu Datentypen und Variablen

Variablennamen können in einer Programmiersprache frei gewählt werden. Am besten so, dass der Name einen eindeutigen Hinweis auf die Art der Daten gibt. Fülle die Tabelle aus, welcher Datentyp sich hinter den folgenden Variablennamen verbergen sollte/könnte und gib ein Beispiel:

_____ PaketAnzahl = _____	_____ istHeuteOstern = _____
_____ SchuelerAnzahl9a = _____	_____ Austragungsort = _____
_____ MeinGewicht_kg = _____	_____ istLuege = _____
_____ Automarke = _____	_____ Guthaben = _____

Die Aufteilung der Ganzzahl-Datentyen in C, bzw. RobotC

Die Programmiersprache C ist eine sogenannte ‚maschinen-nahe' Hochsprache, das bedeutet, dass die Programmiersprache sich sehr eng an der Funktionsweise von Prozessoren und Speichern orientiert. Die Datentypen wie Float oder Integert sind daher so konstruiert, dass der Prozessor sie direkt versteht. Das bedeutet:

Jeder Wert einer Variablen wird im Computerspeicher abgelegt. Wird ein Variablenwert größer, so benötigt er im Computerspeicher ebenfalls mehr ‚Raum': Beispielsweise benötigt ein Ganzzahl-Wert von 127 weniger Speicherplatz als ein Ganzzahl-Wert von 32767.

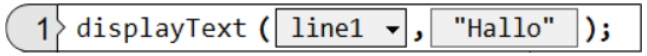
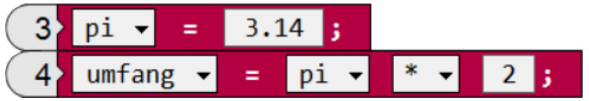
Auf einem Roboter wie dem EV3 ist der Speicherplatz knapp. Deshalb definiert man Variablen so, dass man ihnen nur so viel Speicherplatz wie nötig einräumt. Darüber muss man beim Programmieren nachdenken: Soll zum Beispiel der gemessene Ganzzahlwert des Helligkeitsensors abgespeichert werden, der zwischen den Werten 0 und 100 liegt, wird wenig Computerspeicher dafür benötigt. Will man die Laufzeit des Roboters messen, die üblicherweise in Millisekunden gemessen wird, so beträgt dieser Wert nach einer Stunde bereits 3600 Sekunden * 1000 = 3,6 Millionen. Dafür benötigt man einen größeren Bereich im Computerspeicher.

Ganzzahlen unterscheidet man daher nach ihren Wertebereichen:

Ganzzahl-Datentyp	Wertebereich	Speicherbedarf
byte	-127 bis +127	1 Byte
short	- 32 767 bis + 32 767	2 Byte
int und long	- 2 147 483 648 bis + 2 147 483 647	4 Byte

Anweisungen

Ein Programm besteht aus einzelnen Anweisungen. Jede Anweisung wird vom Computer ausgeführt. Beispiele für Anweisungen in RobotC sind:

Anweisung Graphical RobotC	Anweisung in Text RobotC	Was der EV3-Roboter damit macht
	`displayText(line1, "Hallo");`	Ausgabe von `Hallo Welt` am LCD-Bildschirm des EV3
	`setMotor(motorA, 50);`	Setze Motor A auf 50% Geschwindigkeit
	`float pi = 3.14;` `float umfang = pi * 2;`	Zuweisung des Float-Zahlenwerts 3.14 der Variable mit Namen pi, anschließend Zuweisung des Float-Zahlenwerts 6,28 der Variable umfang

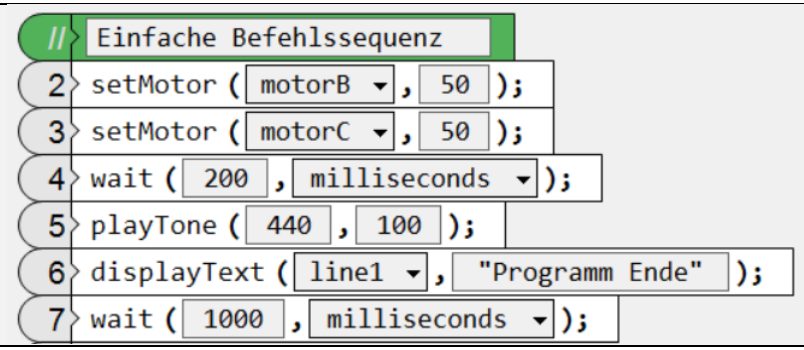
Regel in RobotC:

Eine Anweisung wird in eine Zeile geschrieben. Das Semikolon , ; ' zeigt dem C-Compiler an, dass die Anweisung beendet ist und nun eine neue Anweisung folgt.

Kontrollstruktur Teil 1: Sequenz von Anweisungen

Ein Computerprogramm besteht aus einer Abfolge von Anweisungen, das nennt man Sequenz. Diese Anweisungen werden der Reihe nach, von oben nach unten abgearbeitet. Ab und zu kommen Sprünge vor, in denen das Programm an eine andere Stelle springt. Dort wird wieder sequentiell weiter abgearbeitet.

Graphical RobotC



Text RobotC

```
task main()
{
    // Einfache Befehlssequenz
    setMotor(motorB, 50);
    setMotor(motorC, 50);
    sleep(200);
    playTone(440, 100);
    displayText(line1, "Programm Ende");
    sleep(1000);
}
```

Was das Programm (die Anweisungs-Sequenz) macht, in Zeilen:

1.: mit „//“ wird in RobotC ein Kommentar gekennzeichnet

2.: Motor B wird auf 50% Geschwindigkeit gesetzt

3.: Motor c wird auf 50% Geschwindigkeit gesetzt

4.: Das Programm wartet 0,2 Sekunden

5.: Ein Ton mit 44 Hz wird für 1 Sekunde abgespielt

6.: Auf dem Bildschirm in Zeile 1 der Text „Progamm Ende“ schreiben

7.: Es wird nochmals eine Sekunde gewartet

Regel in RobotC:

Das sogenannte Hauptprogramm wird zu Beginn mit der Anweisung task main() gekennzeichnet und in Klammern geschrieben.

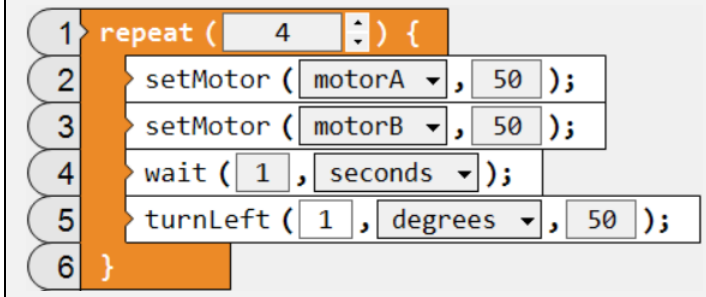
4. Kontrollstruktur II: Schleifen

Will man Anweisungssequenzen häufiger wiederholen (Beispiel: 4-mal „geradeausfahren, dann 90° abbiegen“ um ein Quadrat zu fahren), nutzt man Schleifen.

Zählschleife:

Eine Variable wird bei jedem Schleifendurchlauf hochgezählt bis zum Maximalwert. Ist dieser erreicht, wird die Schleife beendet.

Graphical RobotC



Text RobotC

```
task main()
{
    for(int i=0; i<4; i++) {
        setMotor(motorA, 50);
        setMotor(motorB, 50);
        wait(1, seconds);
        turnLeft(1, degrees, 50);
    }
}
```

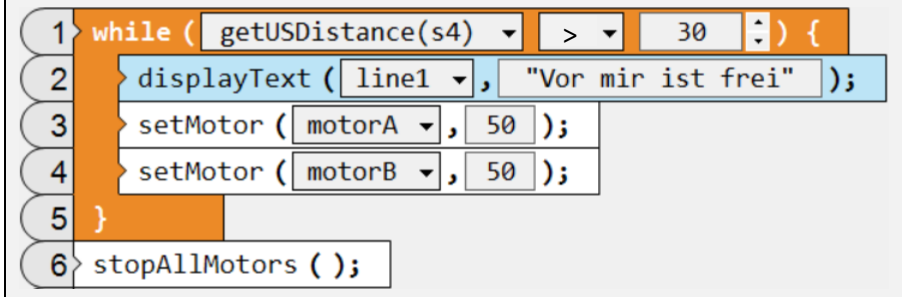
Was das Programm macht:

Die for (bzw. repeat)-Schleife wird 4 mal durchlaufen. In jedem Durchlauf fährt der Roboter mit 50% Geschwindigkeit für 1 Sekunde geradeaus, dann biegt er links ab, indem er sich um 1 Grad dreht.

Bedingungsgesteuerte Schleife

Ob die Schleife wiederholt wird, muss mit einer Bedingung geprüft werden. Ist die Bedingung wahr, wird die Schleife wiederholt; ist die Bedingung falsch, wird die Schleife beendet.

Graphical RobotC



Text RobotC

```
task main()
{
    while (getUSDistance(s4) > 30) {
        displayText(line1, "Vor mir ist frei");
        setMotor(motorA, 50);
        setMotor(motorB, 50);
    }
    stopAllMotors();
}
```

Was das Programm macht:

Solange der Ultraschallsensor einen Abstandswert von mehr als 30 cm erkennt, wird auf das LCD-Display die Information „Vor mir ist frei“ geschrieben und der Roboter fährt geradeaus. Falls das nicht mehr gilt, werden alle Motoren gestoppt.

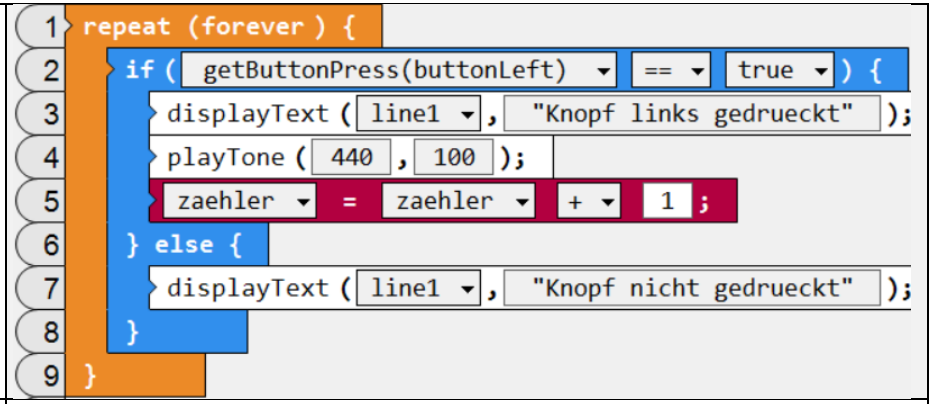
Kontrollstruktur Teil III: Verzweigungen

Wenn zum Beispiel an einem Roboter verschiedene Knöpfe gedrückt werden, soll er jeweils etwas anderes tun. Dazu benötigt das Programm Verzweigungen:

Einfache Verzweigung:

Eine Bedingung wird geprüft. Ist sie wahr, wird Sequenz 1 abgearbeitet, ist sie falsch, wird Sequenz 2 abgearbeitet.

Graphical RobotC



Text RobotC

```
float zaehler = 0;

task main(){
  while (true) {
    if (getButtonPress(buttonLeft) == true) {
      displayText(line1, "Knopf links gedrueckt");
      playTone(440, 100);
      zaehler = zaehler + 1;
    } else {
      displayText(line1, "Knopf nicht gedrueckt");
    }
  }
}
```

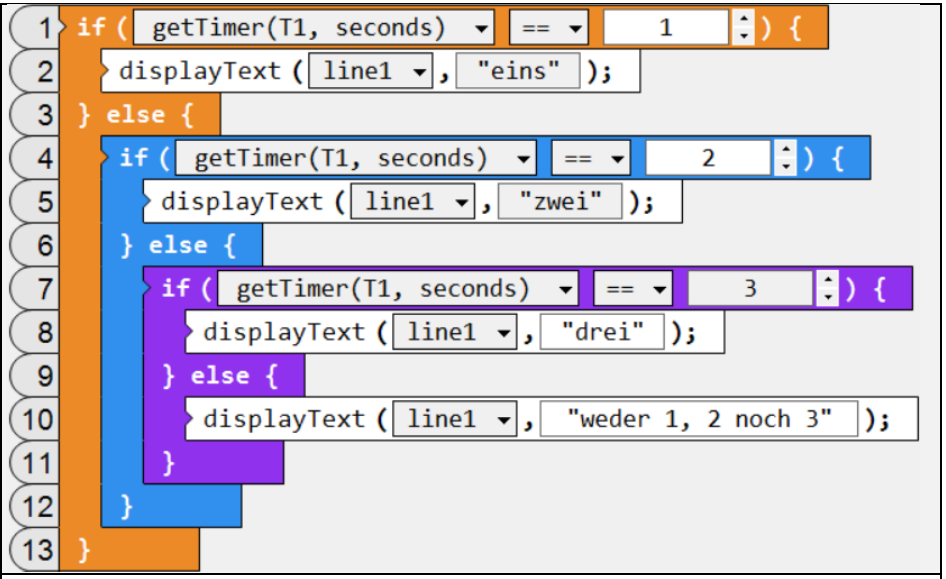
Was das Progamm macht:

Wenn der linke Knopf am EV3 gedrückt wird, wird am LCD-Display der Text „Knopf links gedrückt“ angezeigt und ein Ton wird abgespielt. Wenn der linke Knopf nicht gedrückt wird, liest man am LCD „Knopf nicht gedrueckt“.

Mehrfachverzweigung:

Eine Bedingung wird auf viele unterschiedliche Ergebnisse hin geprüft und dann jeweils eine zugehörige Sequenz abgearbeitet.

Graphical RobotC



Text RobotC

```
task main()
{
    switch (getTimer(T1, seconds)) {
    case 1:     displayText(line1, "eins");break;
    case 2:     displayText(line1, "eins");break;
    case 3:     displayText(line1, "eins");break;
    default:    displayText(line1, "weder 1, 2 noch 3");
    }
}
```

Was das Programm macht:

Der aktuelle Zeitwert des Timers wird abgefragt. Liegt dieser bei 1,2,oder 3 Sekunden, so wird entsprechender Text angezeigt.

```
void allemeine(){
    playTone(262,20);sleep(300);
    playTone(294,20);sleep(300);
    playTone(330,20);sleep(300);
    playTone(349,20);sleep(300);
}

void entchen(){
    playTone(392,40);sleep(600);
    playTone(392,40);sleep(600);
}

void schwimmenADSee(){
    playTone(440,20);sleep(300);
    playTone(440,20);sleep(300);
    playTone(440,20);sleep(300);
    playTone(440,20);sleep(300);
    playTone(392,40);sleep(600);
}

task main(){
  while (true){
    if (getButtonPress(buttonLeft) == true) {allemeine();}
    if (getButtonPress(buttonUp)   == true) {entchen();}
    if (getButtonPress(buttonRight)== true) {schwimmenADSee();}
    if (getButtonPress(buttonDown) == true) {
        allemeine();
        entchen();
        schwimmenADSee();
    }
    sleep(100);
  }
}
```

Kontrollstruktur Teil IV: Funktionen, die Grundlagen

Hier gibt es zur grafischen Programmiersprache von RobotC keine Entsprechungen mehr. Diese Programmstrukturen gibt es nur noch im textbasierten RobotC.

Um häufig wiederkehrende Aufgaben zusammenzufassen, nutzt man Funktionen. Eine Funktion ist eine Anweisungssequenz, die unter einem eigenen Namen abgespeichert wird. Schreibt man nun im normalen Programmablauf den Namen der Funktion, so wird die gesamte darunter abgespeicherte Befehlssequenz abgespielt. Nutzt man Funktionen, erspart dies beim Programmieren viel Zeit und vereinfacht ein Programm erheblich. Funktionen können nämlich immer wieder aufgerufen und wiederverwendet werden.

Was das Programm macht:

Am CPX werden die vier Touchsensoren A1, A2, A3 und A4 angeschaltet.

Es werden drei Funktionen definiert, nämlich

`allemeine(), entchen(), schwimmenAufDemSee()`

Wenn man eine dieser Funktionen aufruft, so spielen sie jeweils eine einprogrammierte Tonsequenz ab.

In der Endlos-while-Schleife werden die Berührungen von A1 bis A4 abgefragt. Beim jeweiliger Berührung wird eine der obigen Funktionen aufgerufen. Lediglich bei A4 werden nacheinander alle Funktionen abgespielt.

Funktionsdefiniton

Man platziert das Schlüsselwort def, dann den Variablennamen, dann ein Klammerpaar und zuletzt einen Doppelpunkt. Nun hat man den Funktionskopf definiert. Im sogenannten Funktionsrumpf wird nun die Anweisungssequenz einprogrammiert. Ruft man im späteren Programm den Rumpf (also den Namen) auf, so wird die jeweilig einprogrammierte Anweisungssequenz abgespielt.

```
void allemeineEntchen(tempo){
    playTone (262,tempo);   sleep(tempo*15);
    playTone (294,tempo);   sleep(tempo*15);
    playTone (330,tempo);   sleep(tempo*15);
    playTone (349,tempo);   sleep(tempo*15);
    playTone (392,tempo*2); sleep(tempo*30);
    playTone (392,tempo*2); sleep(tempo*30);
}

task main(){
  while (true){
    if (getButtonPress(buttonLeft)  == true) {
        allemeineEntchen (10);
    }
    if (getButtonPress(buttonUp)    == true) {
        allemeineEntchen (20);
    }
    if (getButtonPress(buttonRight) == true) {
        allemeineEntchen (40);
    }
    sleep(100);
  }
}
```

Kontrollstruktur Teil V: Funktionen mit Parametern

Einer Funktion kann man Parameter übergeben, die von der Funktion als Variable entgegengenommen werden. Entweder einen oder mehrere Parameter sind möglich.

Was das Programm macht:

Am EV3 werden die drei Buttons links, oben und rechts abgefragt.
Von jedem Knopfdruck wird dieselbe Funktion `allemeineEntchen(tempo)` aufgerufen, jeweils aber mit einem anderen Parameter, den die Funktion unter dem Variablen-Namen `tempo` entgegennimmt. Mit diesem Tempo wird jeweils die Abspielgeschwindigkeit der Tonsequenz bestimmt.

Kontrollstruktur Teil VI: Funktionen mit Rückgabeparameter

Einer Funktion kann nicht nur ein Parameter übergeben werden, eine Funktion kann auch selbst Werte zurückgeben. Diese Rückgabe erfolgt an eine Variable.

Was das Programm macht:

Am EV3 werden die drei Buttons links, oben und rechts abgefragt.
Von jedem Knopfdruck wird dieselbe Funktion `berechneQuadrat(x)` aufgerufen, jeweils aber mit einem anderen Parameter, den die Funktion unter dem Variablen-Namen `x` annimmt. Von diesem Zahlenwert wird das Quadrat berechnet und im Hauptprogramm an die Variable y zurückgegeben. Dann wird im Hauptprogramm der jeweilige y-Wert ausgegeben.

```
int berechneQuadrat(x){
    return x*x
}

task main(){
  while (true){
    if (getButtonPress(buttonLeft)  == true) {
        y = berechneQuadrat(2)
        displayText(line1,"Quadrat von 2: %d", y)
    }
    if (getButtonPress(buttonUp)    == true) {
        y = berechneQuadrat(3)
        displayText(line1,"Quadrat von 3: %d", y)
    }
    if (getButtonPress(buttonRight) == true) {
        y = berechneQuadrat(4)
        displayText(line1," Quadrat von 4: %d", y)
    }
    sleep(100);
  }
}
```