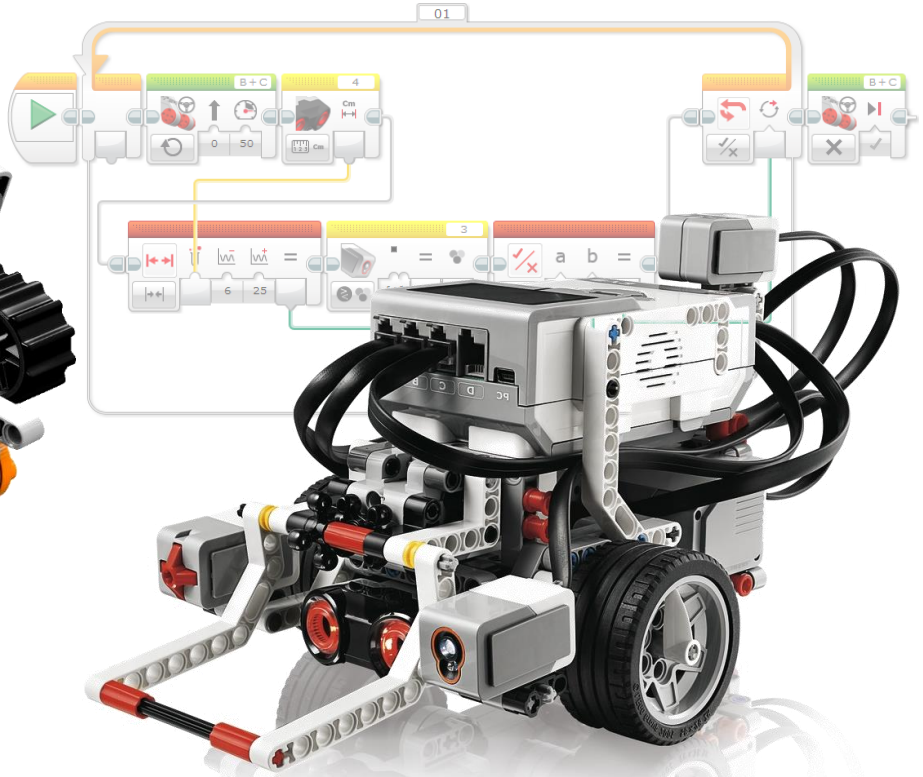
Programmierhandbuch zu NWT Robotics

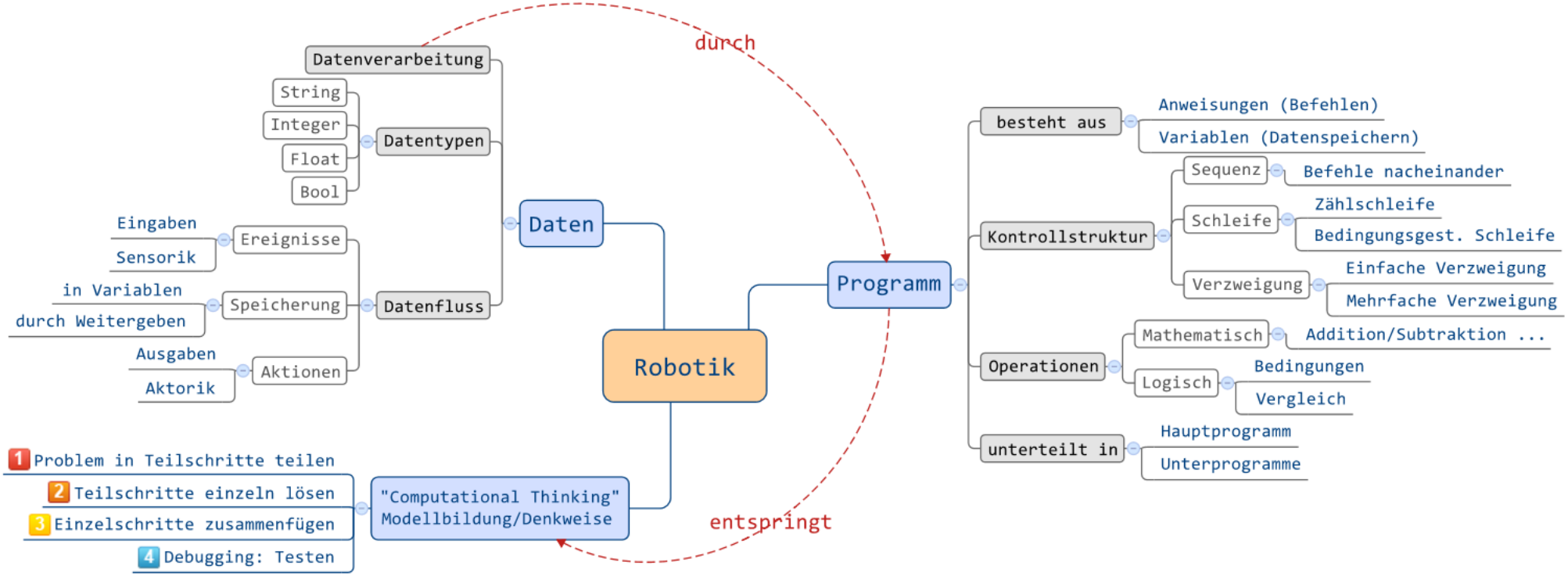
Kepler Gymnasium Weil der Stadt, Version 1.2 (7. März 2017), *Autor: Thomas Jörg*

```
While "True"
  LCD.StopUpdate()
  LCD.Clear()
  LCD.Write(0,0, Sensor.GetName(1))
  LCD.Write(0,16, "MODE: "+Sensor.GetType(1)+":"+Sensor.GetMode(1)+" ("+selectmode+")")
  LCD.Write(0,32, "PERCENT: "+ Sensor.ReadPercent(1))
  raw = Sensor.ReadRaw(1,8)
  For i=0 To 3
    LCD.Write(0,64+i*16, "RAW"+i+":"+raw[i])
  endfor
  For i=4 To 7
    LCD.Write(80,64+(i-4)*16, "RAW"
  endfor
  LCD.Update()

  Program.Delay(200)

  click = Buttons.GetClicks
  If Text.IsSubText(click, "
    selectmode = selectmode+1
    Sensor.SetMode(1, selectmod
  ElseIf Text.IsSubText(click,"D"
    selectmode = selectmod
    Sensor.SetMode(1, s
  ElseIf Text.IsSubTex
    Sensor.SetMode(1, s
  endif
EndWhile
```



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!

1. Kontrollstruktur A: Sequenz von Anweisungen

1.1 Programmelement Hauptprogramm

Ein Computerprogramm besteht aus einer Abfolge von Anweisungen (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 abgearbeitet.

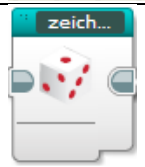
NXT-G	
EV3-G	

NXC	`task main(){`
	`  OnFwdSync(OUT_AB, 50, 0);`
	`  Wait(1000);`
	`  PlayFile("Cat purr.rso");`
	`  GraphicOut(15, 20, "Bomb.ric");`
	`}`

EV3Basic	`Motor.StartSync("AB",50,50)`
	`Program.Delay(1000)`
	`Speaker.Play(100, "Cat purr")`
	`LCD.BmpFile(0,15,20,"Bomb")`

1.2 Programmelement Block / Funktion / Subprogramm

Um häufig wiederkehrende Aufgaben zusammenzufassen, nutzt man Funktionen. Dazu wird eine Anweisungssequenz unter einem Namen abgespeichert. *(Siehe Beispiel „Kurzprogramm“ weiter unten)*

NXT-G	
EV3-G	

NXC	`sub nameDesSubprogramms(){`
	`  //eine Sequenz`
	`  //mit beliebig vielen`
	`  //Anweisungen`
	`}`

EV3Basic	`Sub nameDesSubprogramms`
	`  'Eine Sequenz`
	`  'mit beliebig vielen`
	`  'Anweisungen...`
	`EndSub`

1.2 Aufgabe:

Programmiere eine (sinnvolle) Befehlssequenz und fasse sie in einen eigenen Block / eine eigene Funktion zusammen.
Rufe diesen Block in deinem Hauptprogramm auf.

2. Datentypen: Wie Computer ihre Daten verarbeiten

Grundsätzlich verarbeiten Computer ihre Daten immer in binärer Form, also indem ein Bit (die kleinste Dateneinheit) die zwei Zustände „0“ und „1“ annehmen kann. Aus diesen Bit-Dateneinheiten werden komplexere Datentypen zusammengesetzt. Diese Datentypen entsprechen unseren menschlichen Gewohnheiten:

2.1 int

Ganze Zahlen, negativ und positiv

NXT-G	
EV3-G	
NXC	`int Ganzzahl = 15;`
EV3Basic	`Ganzzahl = 15`

2.2 float

Fließkommazahlen, negativ und positiv

NXT-G	
EV3-G	
NXC	`float Komma = 3.14;`
EV3Basic	`Komma = 3.14`

2.3 string

Buchstaben und Text

NXT-G	
EV3-G	
NXC	`string Txt = „Hallo“;`
EV3Basic	`Txt = "Hallo"`

2.4 boolean

Zwei Logische Werte „True“ & „False“

NXT-G	
EV3-G	
NXC	`bool Boolwert = true;`
EV3Basic	`Boolwert = "True"`

'Bei EV3-Basic gilt: Jede Variable erhält bei der ersten Nutzung einen Datentyp und bleibt dann so.

Wichtig zu wissen: Verarbeitung im Prozessor

Fließkommazahlen und Ganzzahlen werden zwar von einigen Programmiersprachen gleichbehandelt, was aber nicht der Funktionsweise eines Computers entspricht: 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.

3. Variablen: Platzhalter für Daten

Daten können beispielsweise sein: Sensorwerte, Nutzereingaben, Berechnungsergebnisse von Programmen, Lautstärkewerte, Bilddaten, usw. usw. …

3.1 Einfache Variablen zu den einfachen Datentypen

In einem Programm müssen diese Daten gespeichert und aufbewahrt, verarbeitet und geprüft werden. Das geschieht mithilfe von Variablen. Variablen kann man sich als „Behälter“ für Daten vorstellen: Ein Variablen- „Behälter“ speichert einen einzigen Datenwert, und damit man darauf zugreifen kann, erhält der Variablen- „Behälter“ auch einen eigenen Namen. Mit diesem Variablen-Namen kann das Programm jederzeit auf die Variable zugreifen.

Eine Variable kann man sich auch wie in der Mathematik vorstellen: Als Platzhalter für Werte. Ursprünglich stammt der Programmiersprachen-Begriff „Variable“ auch aus der Mathematik (weil viele Informatiker ursprünglich Mathematiker und Physiker waren). In einer mathematischen Variablen wie z.B. „y = 23“ kann y jeden möglichen Wert annehmen. Dabei wäre „y“ der Behälter für den Wert 23. Will man darauf zugreifen, nutzt man den Namen „y“ der Variablen.

Prinzip 1: y = 23 Die Variable mit Namen „y“ bekommt den Wert auf der rechten Seite zugewiesen hier die Ganzzahl „23“, wie auch in einer mathematischen Gleichung.	Prinzip 2: y = y + 1 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***.

3.2 Aufgabe:

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:

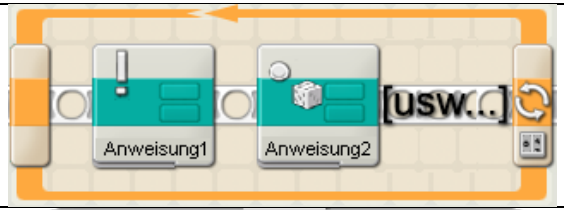
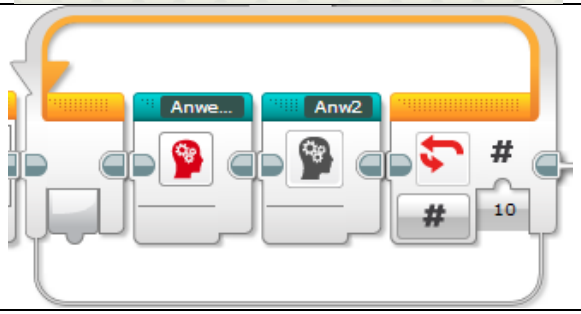
	NXC _____ Paketmenge = _____;		EV3Basic Austragungsort = _____
EV3Basic SchuelerAnzahl9a = _____		EV3Basic istHeuteOstern = _____	
	NXC _____ istLuege = _____;		NXC _____ Guthaben = _____;
NXC _____ Automarke = _____;		EV3Basic MeinGewicht_kg = _____	

4. Kontrollstruktur B: Schleifen

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

4.1 Zählschleife:

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

NXT-G	
EV3-G	

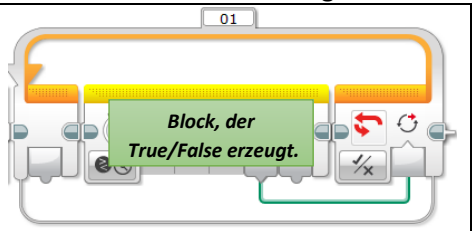
NXC	`for (int i=0;i<10;i++)`
	`{`
	`  //Anweisung1;`
	`  //Anweisung2;`
	`  //usw…`
	`}`

EV3Basic	`For i = 1 To 10`
	`  'Anweisung1`
	`  'Anweisung2`
	`  'usw. ...`
	`EndFor`

4.2 Bedingungsgesteuerte Schleife

Ob die Schleife wiederholt wird, muss mit einer Bedingung geprüft werden.
Eine Bedingung wird entweder zum Beginn der Schleife geprüft, dann ist sie „kopfgesteuert“ oder am Ende, dann ist sie „fußgesteuert“. Ist die Bedingung wahr, wird wiederholt; ist die Bedingung falsch, wird die Schleife beendet.

Die Bedingungsgesteuerten Schleifen von NXT-G und EV3-G sind immer fußgesteuert:

NXT-G		EV3-G	

Die „while“-Schleifen in den textbasierten Sprachen sind meistens kopfgesteuert:

NXC	`while(boolVar == true){`
	`  //Anweisung1;`
	`  //Anweisung2;`
	`  //usw…`
	`}`
		EV3Basic	`While boolVar = "True"`
			`  'Anweisung1`
			`  'Anweisung2`
			`  'usw. ...`
			`EndWhile`

4.3 Aufgabe:

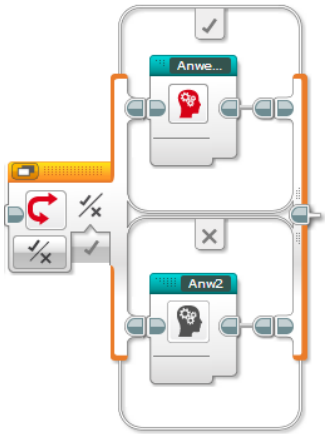
Programmiere deinen Roboter so, dass er solange fährt, bis er 30 cm vor der Wand steht. Dann soll er sich um 180° drehen.
Wenn er das dreimal gemacht hat, soll das Programm enden.

5. Kontrollstruktur C: Verzweigungen

Wenn zum Beispiel ein Roboter irgendwelche Dinge nach Farben sortieren soll, so legt er jede Farbe in einen eigenen Behälter. Zu jeder Farbe tut das Programm des Roboters also etwas Anderes. Um zu verschiedenen Bedingungen (hier z.B. „rot“, „blau“, „gelb“ usw.) eine jeweilige Anweisungssequenz abzuarbeiten, benötigt das Programm Verzweigungen:

5.1 Einfache Verzweigung:

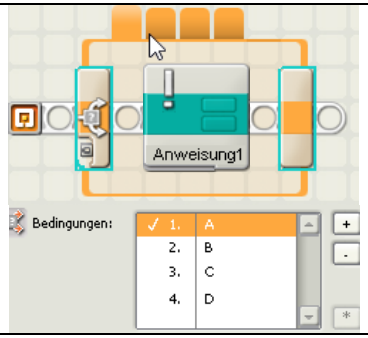
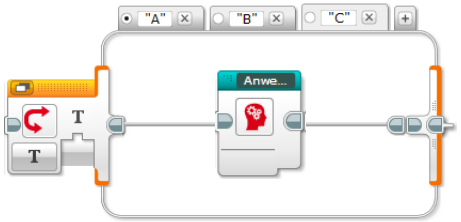
Eine Bedingung wird geprüft. Ist sie wahr, wird Sequenz 1 abgearbeitet, ist sie falsch, wird Sequenz 2 abgearbeitet.

NXT-G		EV3-G	

NXC	`if (x>1){`
	`  //Anweisung1…;`
	`} else {`
	`  //Anweisung2…;`
	`}`
		EV3Basic	`If x>1 Then`
			`  'Anweisung1 ...`
			`Else`
			`  'Anweisung2 ...`
			`EndIf`

5.2 Mehrfachverzweigung:

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

NXT-G		EV3-G	

NXC	`switch(x){`
	`  case „A“:`
	`  //AnweisungenA…;`
	`  break;`
	`  case „B“:`
	`  //AnweisungenB…;`
	`  break;`
	`  case „C“:`
	`  //AnweisungenC…;`
	`  break;`
	`  default:`
	`  //Anweis. sonst…;`
	`  break;}`
		EV3Basic	`If x = "A" Then`
			`  'Anweisungen A ...`
			`ElseIf x = "B" Then`
			`  'Anweisungen B ...`
			`ElseIf x = "C" Then`
			`  'Anweisungen C ...`
			`Else`
			`  'Anweisungen sonst...`
			`EndIf`

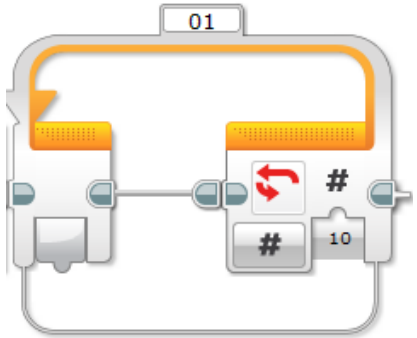
5.3 Aufgabe

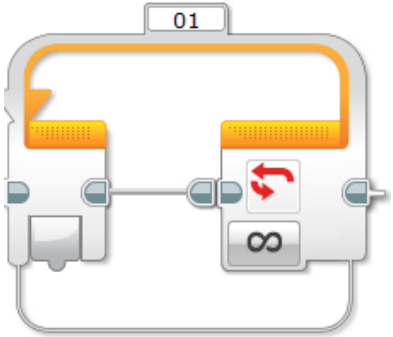
Programmiere einen Roboter so, dass auf dem Bildschirm ausgegeben wird „zu hell“, wenn der Helligkeitswert größer als 50 ist und „zu dunkel“, wenn er kleiner als 50 ist.

Befehlsreferenz für alle vier Programmiersprachen

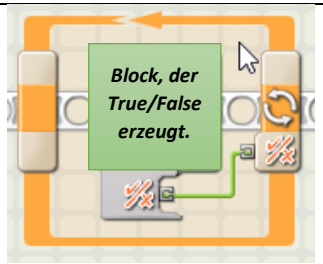
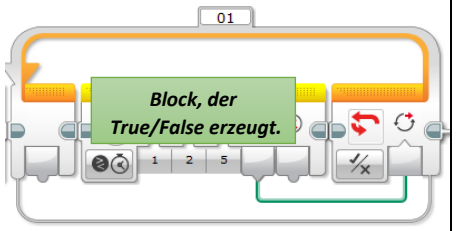
	NXT-G	EV3-G	Bricxcc NXC	EV3Basic

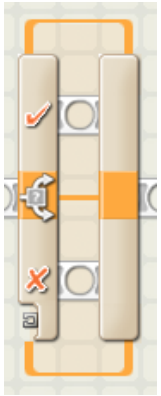
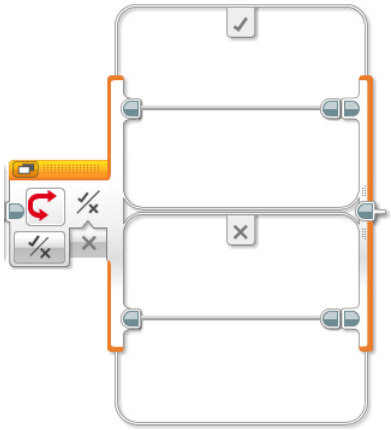
Programmstart			`task main(){`
			`  Anweisungen…`
			`}`
				- Kein Start notwendig -

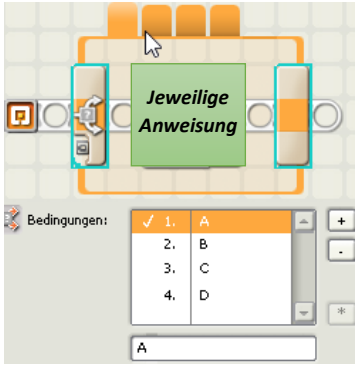
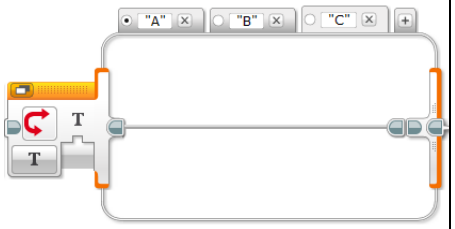
Zählschleife			`for (int i=1; i<11; i++)`
			`{`
			`  Anweisungen…);`
			`}`
			//hier:Variable i von 1 - 10
				`For i = 1 To 10`
				`  'Anweisungen ...`
				`EndFor`
				
				'hier:Variable i läuft von 1 - 10

Endlosschleife			`while (true){`
			`  Anweisungen…`
			`}`
				`While "True"`
				`  'Anweisungen ...`
				`endwhile`

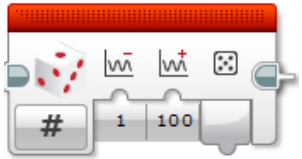
Warten			`Wait(1000);` ***//hier: 1000ms***	`Program.Delay(1000)` *'hier: 1000ms*

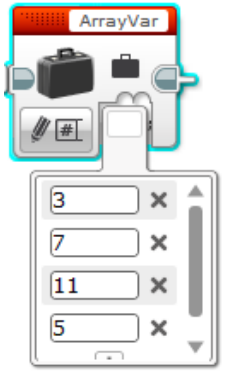
Do-While-Schleife (Fußgesteuert)			`do`
			`{`
			`  Anweisungen…`
			`}while(boolVar = true);`	

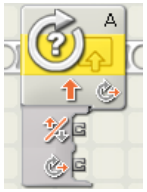
Verzweigung			`if (x>1)`
			`{`
			`  Anweisung1…;`
			`} else {`
			`  Anweisung2…;`
			`}`
				`If x>1 Then`
				`  'Anweisung 1 ...`
				`Else`
				`  'Anweisung 2 ...`
				`EndIf`

Switch-Case	
	Falls „A“, dann…
	Falls “B“, dann … usw.
		
		Falls „A“, dann…
		Falls “B“, dann … usw.
			`switch(x)`
			` {`
			`  case „A“:`
			`  AnweisungenA…;`
			`  break;`
			`  case „B“:`
			`  AnweisungenB…;`
			`  break;`
			`  case „C“:`
			`  AnweisungenC…;`
			`  break;`
			`  default:`
			`  Anweisungen sonst…;`
			`  break;`
			` }`
				`If x = "A" Then`
				`  'Anweisungen A ...`
				`ElseIf x = "B" Then`
				`  'Anweisungen B...`
				`ElseIf x = "C" Then`
				`  'Anweisungen C...`
				`Else`
				`  'Anweisungen ansonsten...`
				`EndIf`

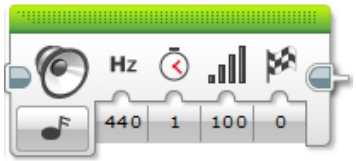
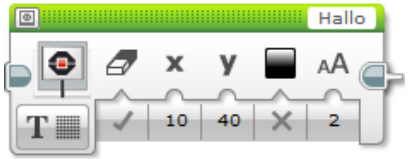
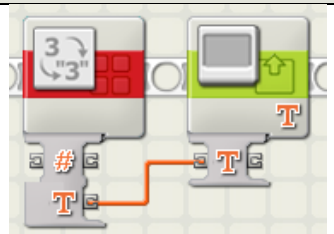
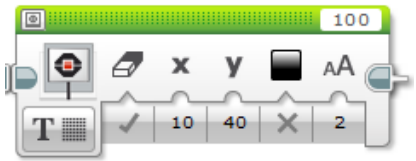
Variable (Zahl)			`int meineVar = 25;`
			`long meineVar = 250000000;`
			`float meineVar = 3.141592;`
				`meineVar = 25`
				`meineVar = 2500000000`
				`meineVar = 3.141592`
				'Jede Variable erhält bei der ersten Nut-
				'zung einen Datentyp und bleibt dann so.

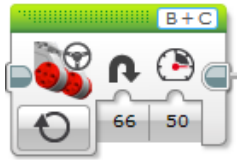
Variable (bool)			`bool logikVar = true;`	`logikVar = "True"` *'Jede Variable erhält bei der ersten Nut-* *'zung einen Datentyp und bleibt dann so.*
Variable (Text)			`string textVar = „Hallo";`	`textVar = "Hallo"` *'Jede Variable erhält bei der ersten Nut-* *'zung einen Datentyp und bleibt dann so.*
Zufallszahl			`Random(100);` ***//Verwendung z.B.:*** ***int i = Random(100);***	`Math.GetRandomNumber(100)` *'Verwendung z.B.:* *'i = Math.GetRandomNumber(100)*

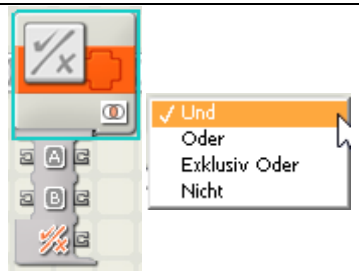
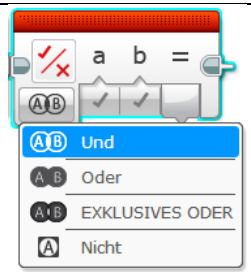
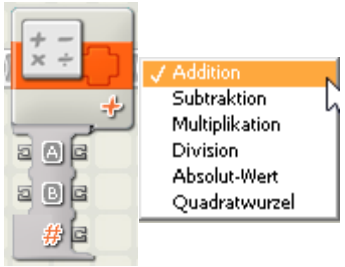
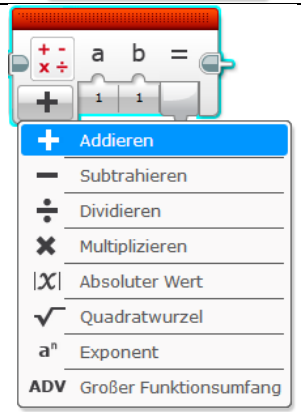
Arrays			`int ArrayVar[] =`
			`{3, 7, 11, 5}`
				`ArrayVar[4]`
				
				`ArrayVar[0] = 3`
				`ArrayVar[1] = 7`
				`ArrayVar[2] = 11`
				`ArrayVar[3] = 5`
				'Jede Variable erhält bei der ersten Nut-
				'zung einen Datentyp und bleibt dann so.

Lichtsensor			SetSensorLight(IN_3); ***//Anmelden zu Beginn*** Sensor(IN_3); ***//Zugriff im Programm***	Sensor.SetMode(3,1) *'Port 3 wird „Licht-An“ geschaltetet.* *'Licht-Aus: 0, Licht-An: 1,* *'Standard Farbe: 2, Full RGB: 4* Sensor.ReadPercent(3) *'kann die Werte 0 bis 100 annehmen*
Tastsensor			SetSensorTouch(IN_1); ***//Anmelden zu Beginn*** Sensor(IN_1); ***//Zugriff im Programm***	Sensor.ReadPercent(1) *'kann die Werte 0 ODER 100 annehmen*
Tonsensor			SetSensorSound(IN_2); Sensor(IN_2); ***//Zugriff im Programm***	Sensor.ReadPercent(2) *'kann die Werte 0 bis 100 annehmen*
Ultraschall-Sensor			SetSensorLowspeed(IN_4); ***//Anmelden zu Beginn*** SensorUS(IN_4); ***//Zugriff im Programm***	Sensor.ReadRawValue(3,0) *'Ultraschallsensor in Port 3;* *'Beim zurückgegebenen Array wird der* *'Wert mit dem Index 0 genutzt:* *'gemessener Abstand in mm*
Motor-Drehsensor			***//gibt den Gesamtdrehwinkel*** MotorRotationCount(OUT_A);	Motor.GetCount(„A“)
Tasten am Brick			***//gibt True od. False zurück*** ButtonPressed(BTN1,false); ButtonPressed(BTN2,false); ButtonPressed(BTN3,false); ButtonPressed(BTN4,false);	Buttons.Wait()*'wartet bis Tastendruck* Buttons.GetClicks()*'speichert ab* Buttons.GetCurrent()*'nur aktueller* *'mögl. Werte: „U“, „D“, „L“, „R“, „E“* Buttons.Flush()*'Buttonclicks löschen*

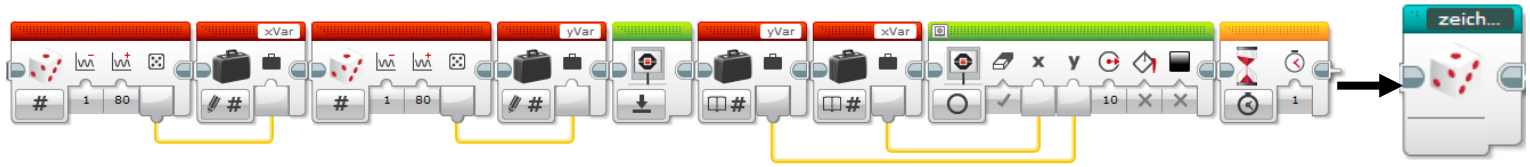
Timer-Sensoren des Microcontrollers	
	3 verschiedene Timer!
		
		8 verschiedene Timer!
			`long zeitVar;`
			`zeitVar = CurrentTick();`
			
			//wird in Millisekunden ab
			//Programmstart gemessen
				`zeitVar = EV3.Time`
				'wird in Millisekunden ab
				'Programmstart gemessen

Ton abspielen			`PlayToneEx(440,90,50,FALSE);` ***//Frequency 440 Hertz,*** ***//duration 90ms,*** ***//volume 100,*** ***//loop FALSE***	`Speaker.Tone (100, 440, 1000)` *'volume 100,* *'frequency 440,* *'duration 1000ms*
LCD Text			`TextOut(10, 40, „Hallo");` ***//Koord.: x: 10px, y: 40 Px*** ***//anstelle y geht auch*** ***//LCD_LINE1 … LCD_LINE8***	`LCD.Write(10,40,"Hallo")` *'x-Koordinate 10 Pixel,* *'y-Koordinate 40 Pixel*
LCD Zahl			`NumOut(10, 40, 100);` ***//Koord.: x: 10px, y: 40 Px*** ***//anstelle y geht auch*** ***//LCD_LINE1 … LCD_LINE8***	`LCD.Write(10,40, 100)` *'x-Koordinate 10 Pixel,* *'y-Koordinate 40 Pixel*
LCD Clear			`ClearScreen();`	`LCD.Clear()`

Motor(en) vorwärts			`OnFwd(OUT_C, 50);` ***//OUT_A, OUT_B, OUT_C*** ***//OUT_AB, OUT_AC, OUT_BC***	`Motor.Start("C",50)` *'„A“, „B“, „C“, „AB“, „AC“, „BC“*
Motor(en) rückwärts			`OnRev(OUT_C, 50);` ***//OUT_A, OUT_B, OUT_C*** ***//OUT_AB, OUT_AC, OUT_BC***	`Motor.Start("C",-50)` *'„A“, „B“, „C“, „AB“, „AC“, „BC“*
Motor(en) aus			`Off(OUT_C);` ***//OUT_A, OUT_B, OUT_C*** ***//OUT_AB, OUT_AC, OUT_BC***	`Motor.Stop("C", "True")` *'„A“, „B“, „C“, „AB“, „AC“, „BC“*
Synchrone Motoren			`OnFwdSync(OUT_BC, 50, 66);` ***//OUT_AB, OUT_AC, OUT_BC***	`Motor.StartSync("AC",75,25)` *' „AB“, „AC“, „BC“ ACHTUNG! hier:* *'75 ist Speed von A, 25 Speed von B*
Messe Leistung	- Nicht möglich -		`x = MotorPower(OUT_A);` ***//OUT_A, OUT_B, OUT_C***	`Motor.GetSpeed("A")` *'„A“, „B“, „C“*
Motor Reset			***//setzt Wert auf 0*** `ResetRotationCount(OUT_A);`	`Motor.ResetCount("A")`

Logik-Operationen			A && B (logisches UND) A || B (logisches ODER) ***//exklusiv-Oder gibt's nicht*** !A (logisches NICHT)	A and B (logisches UND) A or B (logisches ODER) *'exklusiv-Oder gibt's nicht* *'logisches NICHT gibt's nicht*
Mathematik-Operationen			A + B (Addition) A – B (Subtraktion) A * B (Multiplikation) A / B (Division) Abs(A) (Absolutwert) Sqrt(A) (Quadratwurzel) pow(A,3) (Exponent)	A + B A - B A * B A / B Math.Abs(A) Math.SquareRoot(A) Math.Power(A,3)
Vergleichs -Operationen			A ==B (gleich) A !=B (ungleich) A > B (größer als) A >=B (größer gleich) A < B (kleiner als) A <=B (kleiner gleich)	A = B (gleich) A <>B (ungleich) A > B (größer als) A >=B (größer gleich) A < B (kleiner als) A <=B (kleiner gleich)
Text -Operationen	Gibt es nicht		„Er hat "+ Anzahl + „Aepfel" ***//Es gibt sehr, sehr viele*** ***//weitere String-Operationen***	„Er hat " + Anzahl + „Aepfel" *'Es gibt sehr, sehr viele* *'weitere String-Operationen*

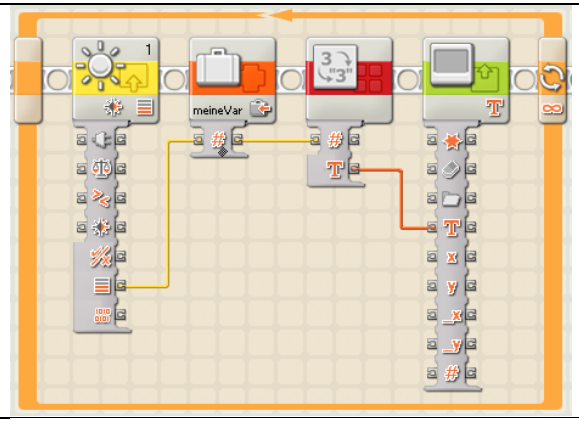
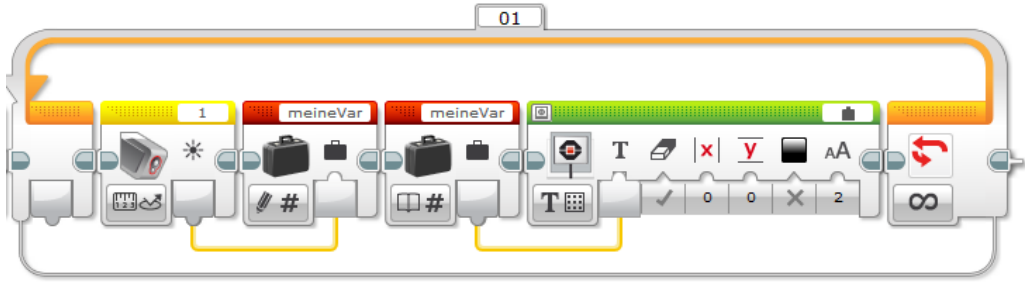
Kurzprogramm 1: Eine Funktion bzw. Subroutine erstellen *anhand des Beispiels „Zufallkreis auf Display zeichnen“:*

NXT-G				· Befehlssequenz auswählen, · dann im Menü „Bearbeiten“: · · „Neuen eig. Block erstellen“ · der Menüführung folgen. · im Hauptprogramm einbauen.
EV3-G				· Befehlssequenz auswählen, · dann im Menü „Werkzeuge“: · „Eigene Blöcke erstellen“ · der Menüführung folgen · im Hauptprogramm einbauen.

NXC	`int xVar = Random(80);`
	`int yVar = Random(80);`
	`ClearScreen();`
	`CircleOut(xVar,yVar,10);`
	`Wait(1000);`
		`sub zeichneZufallK(){`
		`int xVar = Random(80);`
		`int yVar = Random(80);`
		`ClearScreen();`
		`CircleOut(xVar,yVar,10);`
		`Wait(1000);`
		`}`
			`task main(){`
			`zeichneZufallK();`
			`}`
				· Befehlssequenz erstellen,
				· in einen „Sub“-Block fügen
				· dem Sub-Block Namen geben
				· im Main-Task aufrufen

EV3Basic	`xVar = Math.GetRandomNumber(80)`
	`yVar = Math.GetRandomNumber(80)`
	`LCD.Clear()`
	`LCD.Circle(1,xVar,yVar,10)`
	`Program.Delay(1000)`
		`Sub zeichneZufallsK`
		`xVar = Math.GetRandomNumber(80)`
		`yVar = Math.GetRandomNumber(80)`
		`LCD.Clear()`
		`LCD.Circle(1,xVar,yVar,10)`
		`Program.Delay(1000)`
		`EndSub`
			`zeichneZufallsK ()`	· Befehlssequenz erstellen,
				· in einen „Sub“-Block fügen
				· dem Sub-Block Namen geben
				· im Hauptprogramm aufrufen

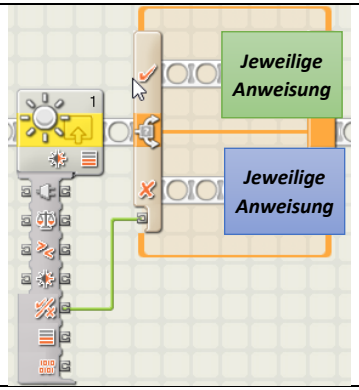
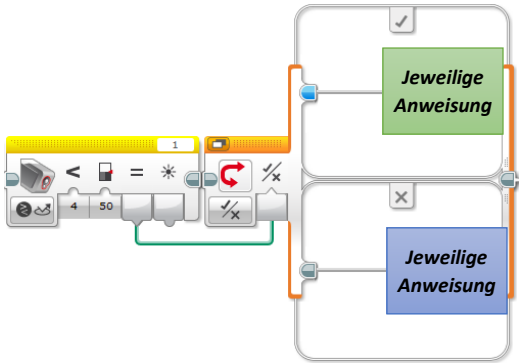
Kurzprogramm 2: Helligkeitswert auf dem Display darstellen

NXT-G		- Helligkeitssensor auslesen - in Zahlenvariable schreiben (hier: meineVar) - Zahlenvariable in Text konvertieren („Spezialität“ NXT-G) - Text auf dem Display ausgeben - Sequenz in einer Schleife von Vorne
EV3-G		- Helligkeitssensor auslesen - in Zahlenvariable schreiben (hier: meineVar) - Zahlenvariable auf dem Display ausgeben - Sequenz in einer Schleife von Vorne

NXC	`int meineVar;`
	`SetSensorLight(IN_1);`
	`while (true){`
	`  meineVar = SENSOR_1;`
	`  NumOut(0, LCD_LINE1, meineVar);`
	`}`
		- Integervariable meineVar deklarieren
		- Lichtsensor auf Port 1 deklarieren
		- Endlosschleife:
		- „meineVar“ wird der aktuelle Sensorwert zugewiesen
		- „meineVar“ wird auf dem Bildschirm ausgegeben

EV3Basic	`While "True"`
	`  meineVar = Sensor.ReadPercent(1)`
	`  LCD.Clear()`
	`  LCD.Write(50,50, prozent)`
	`endwhile`
		- Endlosschleife:
		- Variable meineVar erhält Sensorwert in % von Port 1
		- LCD wird gelöscht
		- Variable meineVar wird auf dem Bildschirm ausgegeben

Kurzprogramm 3: Verzweigung anhand eines Helligkeitswertes

NXT-G		Der Logik-Ausgang des Lichtsensorblocks wird mit dem Eingang der Verzweigung gekoppelt.
EV3-G		Der Logik-Ausgang des Lichtsensorblocks wird mit dem Eingang der Verzweigung gekoppelt.

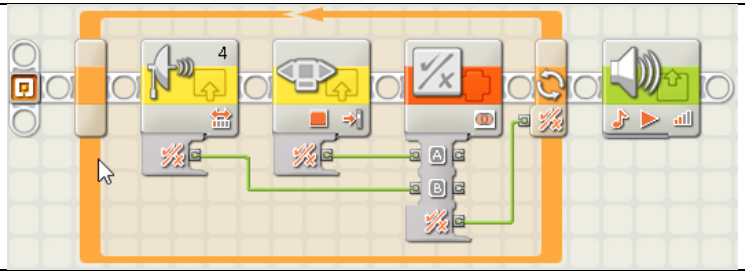
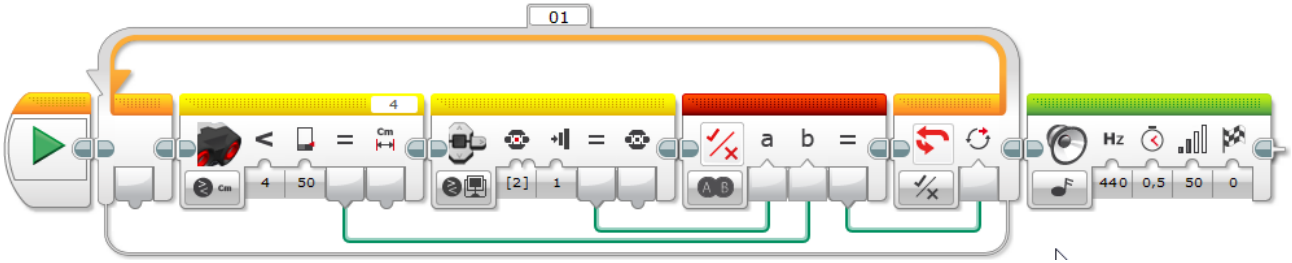
NXC	`SetSensorLight(IN_1);`
	`if (SENSOR_1<50){`
	`  Anweisung1…;`
	`} else {`
	`  Anweisung2…;}`
		Lichtsensor wird an Port 1 angemeldet,
		Sensorwert an Port 1 wird ausgelesen,
		Bedingung: Ist der Wert grösser 50 …
		… dann erfolgen die Anweisungen 1
		… ansonsten die Anweisungen 2

EV3Basic	`If Sensor.ReadPercent(1)<50 Then`
	`  'Anweisungen 1...`
	`Else`
	`  'Anweisungen 2...`
	`EndIf`
		Sensorwert an Port 1 wird ausgelesen,
		Bedingung: Ist der Wert grösser 50 …
		… dann erfolgen die Anweisungen 1
		… ansonsten die Anweisungen 2

Kurzprogramm 4: Logische Verknüpfung von Bedingungen

Innerhalb der Schleife wird permanent gemessen:

- Ultraschallsensor misst Entfernung größer 50: ergibt „True“ ,
- die mittlere Taste wird **nicht** gedrückt: ergibt „True“.
- Sind beide „True“, berechnet die Oder-Operation: ergibt „True“ und die Schleife wird wiederholt.
  - Ist eines „False“, berechnet die Oder-Operation: ergibt „False“ und die Schleife wird unterbrochen, ein Ton abgespielt und das Programm beendet.

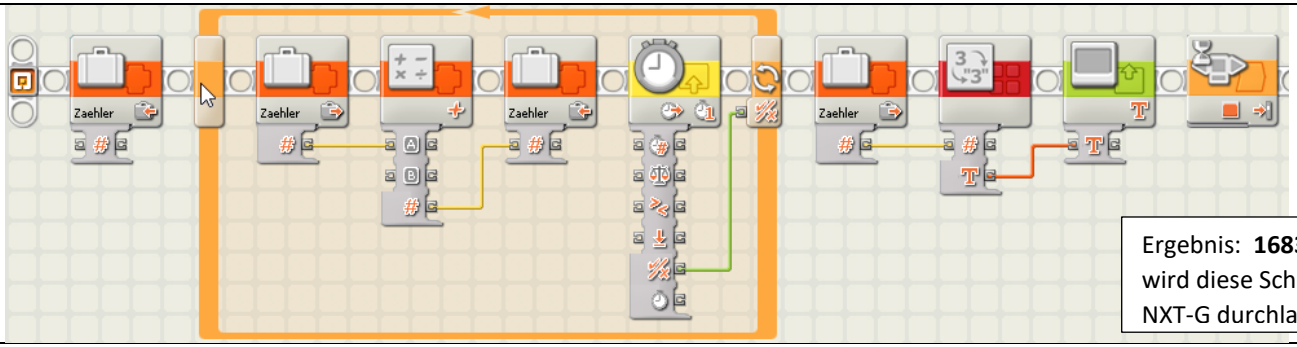
NXT-G		
EV3-G		Wie oben.

NXC	`task main(){`
	`  SetSensorLowspeed(IN_4);`
	`  while(SensorUS(IN_4)>50 || ButtonPressed(BTNCENTER, false)==false);`
	`  PlayToneEx(440, 500, 50, FALSE);`
	`}`
		Ultraschallsensor wird an Port 4 angemeldet, ansonsten wie oben.

EV3Basic	`While (Sensor.ReadRawValue(4,0)>50 And Buttons.Current<>"E")`
	`EndWhile`
	`Speaker.Tone (50, 440, 500)`
		Wie oben.

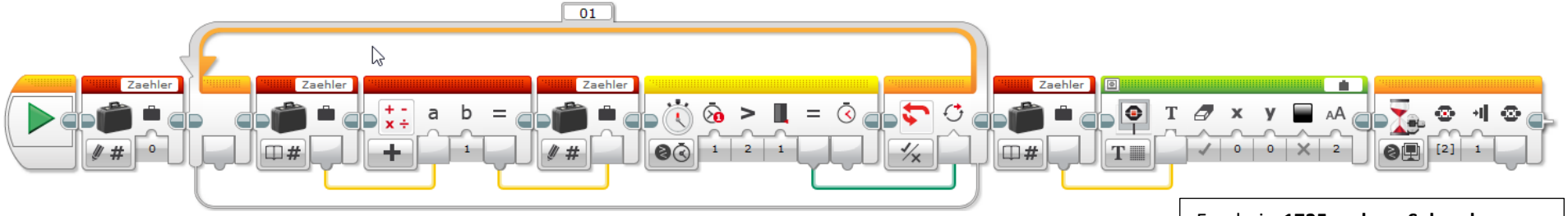
Kurzprogramm 5, nützlich zu wissen: *Wie schnell kann eigentlich eine Schleife wiederholt werden*?

NXT-G



Ergebnis: **1683 mal pro Sekunde** wird diese Schleife bei einem NXT mit NXT-G durchlaufen.

EV3-G



Ergebnis: **1735 mal pro Sekunde** wird diese Schleife bei einem EV3 mit EV3-G durchlaufen.

NXC

```
task main(){
      int Zaehler = 0;
      int Startzeit = CurrentTick();
      int AktuelleZeit = 0;
      while(AktuelleZeit<=1000){
            AktuelleZeit = CurrentTick()-Startzeit;
            Zaehler = Zaehler + 1;
      }
      ClearScreen();
      NumOut(0, 0, Zaehler);
      while(ButtonPressed(BTNCENTER, false)==false);
}
```

Ergebnis: **<u>9525</u> (!) mal pro Sekunde** wird diese Schleife bei einem NXT mit NXC durchlaufen.

EV3Basic

```
Zaehler = 0
StartZeit = EV3.Time
AktuelleZeit = 0
While(AktuelleZeit<=1000)
  AktuelleZeit = EV3.Time - StartZeit
  Zaehler = Zaehler + 1
EndWhile
LCD.Clear()
Buttons.Flush()
LCD.Write(0,0,Zaehler)
Buttons.Wait()
```

Ergebnis: **1852 mal pro Sekunde** wird diese Schleife bei einem EV3 mit EV3Basic durchlaufen.