INFORMATIK MAGIE

Kreativer Einstieg in die faszinierende Welt des Programmierens

hep

Thomas Garaio

Verzeichnis

5 Lernen am Beispiel

Wir beginnen unsere Reise in die Welt der Informatik mit vielen Anwendungsbeispielen. Wir werden also schnell sehen, was wir alles programmieren können, und nachher entdecken, wie es genau funktioniert. Die präsentierten Programme werden einige Überraschungen für uns bereithalten, dennoch können wir die Geheimnisse der Programme lüften.

Das liegt zum einen daran, dass sie einfach und kompakt sind. Die meisten Programme bestehen nur aus wenigen Zeilen Code. Zum anderen liegt es aber auch an den Effekten der Programme. Wir können nämlich direkt sehen, was passiert, wenn wir das Programm laufen lassen. Die Codes wurden von erfahrenen Entwickler*innen umgesetzt und enthalten auch viele Tipps und Tricks, die uns den Einstieg erleichtern.

Gerade wenn man noch wenig Programmiererfahrung hat, kann der Einstieg in die Informatik sehr holprig sein. Viele Programmierkurse ähneln auch eher einem Mathematiklehrgang, bei dem unendlich lange Grundwissen gepaukt werden muss, bevor man an die interessanten Dinge gelangt. Leider ist es auch so, dass gute Programmierer*innen viel Zeit in ihre Berufung investieren müssen, daher gelingen beeindruckende Codes eigentlich auch erst, wenn man eine gewisse Routine aufgebaut hat.

Dieses Kapitel verfolgt einen anderen Weg: Wir starten mit fertigen Beispielen. Diese können sofort runtergeladen und ausprobiert werden. Durch das integrierte Debugging lassen sich die Programme betrachten und untersuchen und mit dem einfach zu bedienenden Konstanteneditor kann man die Programme sogar anpassen, ohne direkt in den Code einzugreifen. So lernst du viele Konzepte und eignest dir die Grundlagen an, indem du den Code an gezielten Stellen abänderst.

Jedes der Beispiele wird mit komplett dokumentiertem Source-Code geliefert, sodass Interessierte immer auch direkt den Code anschauen können, um zu sehen, wie etwas im Detail funktioniert. Die Kapitel schliessen ab mit kleinen Experimenten, die man unter Anleitung umsetzen kann.

Wenn du diese Beispiele durchgearbeitet hast, wirst du viele Konzepte der Informatik besser verstanden haben. Wir werden alle wesentlichen Teile einer Programmiersprache mehrfach betrachten, u. a. Grundkonstrukte wie Variablen, Datentypen, Listen, Schleifen, Bedingungen, Funktionen und Klassen. Wir werden uns aber auch mit Computergrafik im Allgemeinen, Zufallszahlen und Rauschen, 3D, Kinematik, Vektorgeometrie, Gamedesign und natürlich Algebra beschäftigen, da die Informatik nie zum Selbstzweck betrieben wird. Die Anwendung macht es auch erst fassbar und spannend.

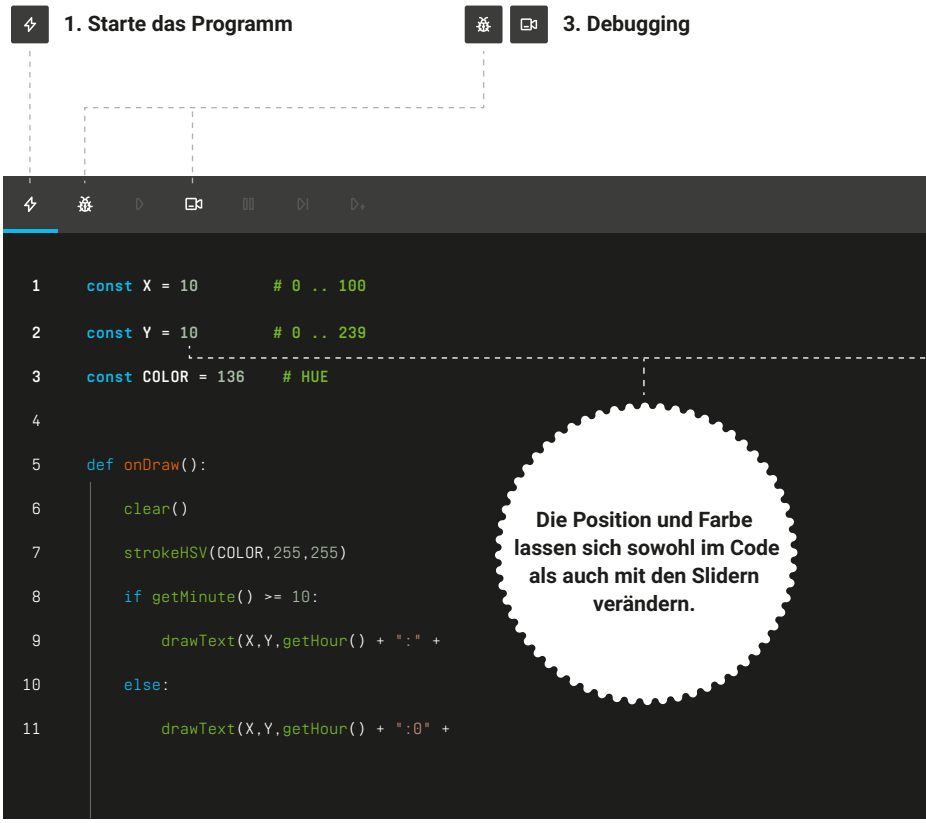
Am Schluss werden bei jedem Beispiel die wichtigsten Punkte unter «Wichtig zu wissen» zusammengefasst.

5.1 Zeit: Digital-Uhr

Dieses Programm stellt die aktuelle Zeit in digitaler Form dar.

Es ist kaum vorstellbar, ein Leben ohne Uhr zu führen. Alle unsere Aktivitäten sind getaktet durch Zeitpunkte, die bestimmen, wann wir aufstehen, den Zug nehmen, Pause machen, Freunde treffen und wann der letzte Zeitpunkt ist, ins Bett zu gehen. Es liegt daher auf der Hand, sich zuerst mit der Zeit zu beschäftigen.

Eine Uhr gibt die Zeit an. Dazu benutzen wir in der Computerwelt meistens Zahlen. Unser Programm ermittelt dabei die Anzahl Sekunden, die seit dem 1.1.1970 00:00:00 in Greenwich (England) vergangen sind. Damit können wir jedoch nur wenig anfangen. Diese Anzahl wird dann mit einem Algorithmus in Jahre, Tage, Stunden und Minuten umgerechnet. Dazu berücksichtigt der Algorithmus auch unsere Zeitzonen und Sommer- und Winterzeit.



5.1.1 Ausprobieren

1. Starte das Programm
Starte das Programm **"Digital watch"** und beobachte die Ausgabe.

2. Experimentiere mit den Konstanten
Ändere nur die Konstante **X** ab – erhöhe diese um Werte zwischen **10** und **100**. Was ändert sich im Programm?

Setzte **X** wieder auf **10** und ändere stattdessen **Y** ab. Was beobachtest du jetzt?

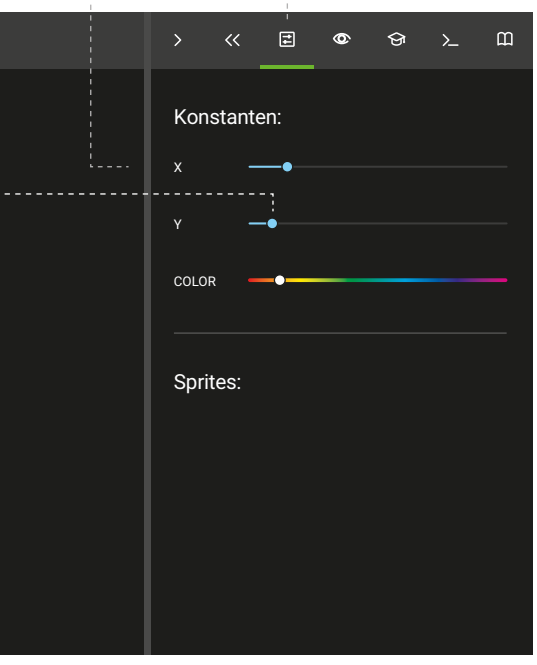
Ändere die Farbe ab. Was bewirkt diese Änderung im Code?

3. Debugging
Beobachte, wie sich das Programm immer wieder von Neuem startet.

Wenn die aktuelle Minutenzahl unter 10 ist, wird die Zeile 11 ausgeführt.

Wenn die aktuelle Minutenzahl 10 oder darüber ist, wird stattdessen die Zeile 9 ausgeführt.

2. Experimentiere mit den Konstanten



X
Verschiebt die horizontale Position der Anzeige (0 = linker Rand, 239 = rechter Rand).

Y
Verschiebt die vertikale Position des Textes (0 = oberer Rand, 239 = unterer Rand).

COLOR
Farbe (0-255, entspricht einer Farbe im HSV-Farbkreis, siehe auch Farbräume in der Online-Dokumentation)

5.1.2 Wie funktioniert es?

Dieses kurze Programm enthält eine Ereignis-Schleife. Dies bedeutet, dass das Programm läuft, solange die Oxocard mit Strom betrieben ist. Man sagt auch: **«Es terminiert nie»**. Die Schleife bekommt die Zeitangaben aus dem Internet und zeigt diese dann auf dem Bildschirm an. Die aktuelle Stunde und die aktuelle Minute sind jeweils simple Zahlen, die noch etwas formatiert werden müssen, damit sie lesbarer sind. Wir haben uns dafür entschieden, zwischen der Stunde und der Minute einen Doppelpunkt zu zeichnen. Zudem schreiben wir vor Minuten unter 10 noch eine vorangestellte **«0»**, damit beispielweise 10:05 anstelle von 10:5 steht.

```
1  const X = 10         # 0 .. 100
2  const Y = 10         # 0 .. 239
3  const COLOR = 39     # HUE
4
5  def onDraw():
6      clear()
7      strokeHSV(COLOR,255,255)
8      if getMinute() >= 10:
9          drawText(X,Y,getHour() + ":" + getMinute())
10     else:
11         drawText(X,Y,getHour() + ":0" + getMinute())
12     update()
13     delay(1000)
```

`1-3` Hier werden die Konstanten definiert, die wir im Konstanteneditor finden. Eine Änderung im Konstanteneditor führt automatisch auch zu einer Änderung im Code.

`5` Die `onDraw-Funktion` ruft die Oxocard immer wieder auf und führt dabei die eingerückten Befehle aus.

`6` Löscht den Bildschirm.

`7` Bestimmt die Strichfarbe, d. h. die Farbe, in der die Zahlen gezeichnet werden.

`8` Wir fragen hier ab, ob die aktuelle Minute grösser 10 ist. Wenn das der Fall ist, können wir die Minuten direkt ausgeben.

`9` Zeichnet die Stunden und Minuten mit einem Doppelpunkt dazwischen. Die einzelnen Teile werden mit + aneinandergefügt. Dieser Block ist eingerückt und bedeutet, dass er nur ausgeführt wird, wenn die Bedingung der Zeile 8 erfüllt ist.

`10` Falls die Bedingung der Zeile 8 nicht erfüllt ist, kommt der folgende Block zum Einsatz.

`11` Wir zeichnen den Text und fügen aber noch eine "0" vor die Minutenanzahl. Dies, weil bei Zahlen unter 10 sonst bloss einstellige Minutenwerte angezeigt würden (z. B. 10:3), was ungewohnt ist.

`12` Dies ist die wichtigste Funktion: Sie stellt erst sicher, dass die Zeichnungsbefehle auf dem Bildschirm angezeigt werden.

`13` `delay` wartet eine Weile, bis die Funktion `onDraw` erneut aufgerufen wird. Die Angabe ist in Millisekunden, 1000 ms = 1 s.

5.1.3 Experimentieren

1. Gestaltung der Uhr verändern
Nutze hierzu die folgenden Befehle:
- `stroke`
- `background`
- `drawLine`
- `drawRectangle`

Diese Befehle können in der **Online-Dokumentation*** nachgeschlagen werden, wo sich auch kleine Beispiele finden lassen.

Tipps: Ersetze `strokeHSV` durch `stroke`, ersetze `clear` durch `background`. Diese Funktionen haben andere Daten / Parameter, die man angeben muss. In der Dokumentation finden sich jeweils Beispiele dazu, die man kopieren kann.

Hier sind ein paar Beispiele:
– Ersetze `clear()` durch `background(255,0,0)`
– Ersetze `strokeHSV...` durch `stroke(0,255,0)`
– Ergänze vor der update-Zeile: `drawLine(1,1,240,1)`

---

2. Sekunden hinzufügen
Die Sekunden sollen hinter den Minuten in Klammern angezeigt werden.

Die folgende Anweisung schreibt die Anzahl Sekunden in runde Klammern:

```
" (" + getSecond() + ")"
```

Wo überall müsste dies im Programm ergänzt werden?

---

3. Text-Variablen nutzen
Man kann den Ausgabetext auch mit einer Text-Variablen zusammensetzen. Hierzu ein Beispiel:

```
s = ""
s = "Es ist " + getHour() + " Uhr"

drawText(10,10,s)
update()
```

Schreib das Programm so um, sodass nur noch eine `drawText`-Zeile notwendig ist.

---

4. Zeitzonen
Mit dem Befehl `setTimeZone` kann eine andere Zeitzone gewählt werden. Die Zeit wird normalerweise in GMT (Greenwich Mean Time) ausgegeben und hat weder Zeitzonen noch Sommer-/Winterzeit berücksichtigt.
Recherchiere den Eintrag in der **Online-Dokumentation*** und ergänze das Programm so, dass abwechselnd die Zeit von Shanghai und Tokyo ausgegeben wird.

5.1.4 Weiterführende Informationen

6.16	Konstanten- und Konstanteneditor
6.9	Bedingungen
6.11.1	Ereignisprozeduren
6.14.2	Texte (Strings)
8.5	Datum und Zeit
8.3.2	Koordinatensystem
8.3.5	Farben
8.3.1	Text ausgeben

5.1.5 Wichtig zu wissen

Computer beziehen die Zeit häufig als eine Anzahl Sekunden über das Internet. Das anzuzeigen, wäre für uns nicht zweckmässig. Deswegen muss sie noch umgerechnet werden.

Wenn die Ereignisprozedur `onDraw` definiert ist, wird diese durch die Oxocard immer wieder automatisch aufgerufen.

`update` bringt die Zeichnungen erst auf den Bildschirm und muss immer am Schluss aufgerufen werden.

Beim Umformatieren von Daten, kann man mit Bedingungen Weichen stellen und die Ausgaben entsprechend steuern.

Mit Zeichnungsbefehlen kann man unter anderem auch Texte in verschiedenen Farben ausgeben.

*Online-Dokumentation

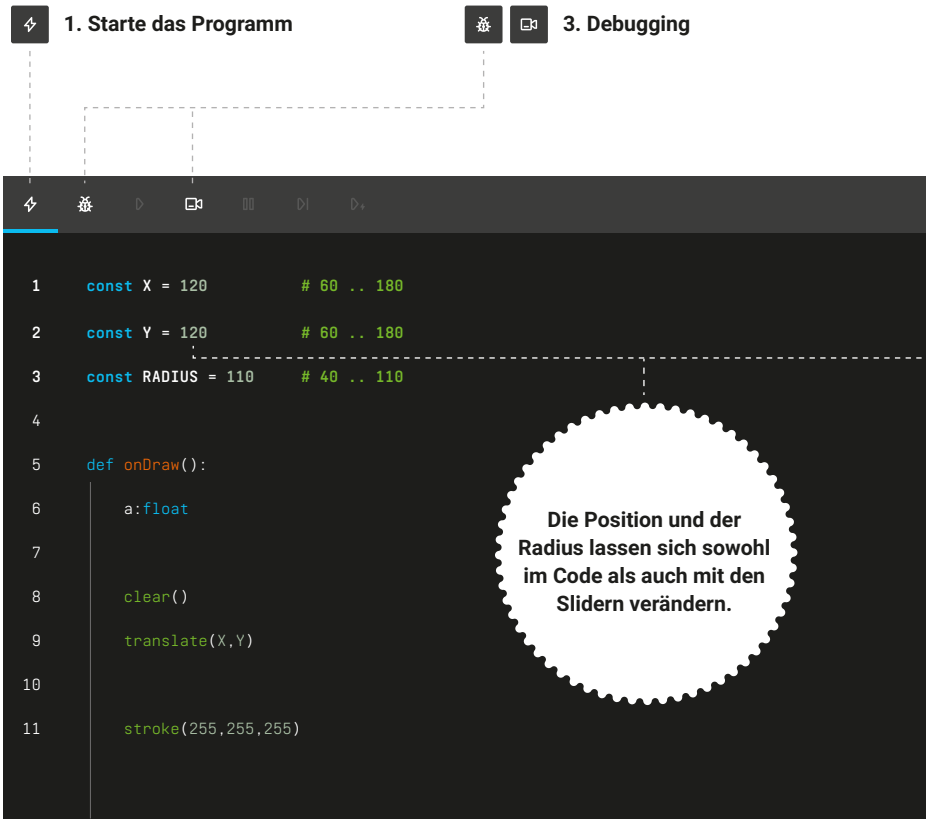
5.2 Zeit: Analog-Uhr

In diesem Beispiel zeichnen wir eine Uhr mit Stunden-, Minuten- und Sekundenzeiger.

Eine klassische Uhr verwendet Zeiger, die um eine Mittelachse drehen. In unserem Fall benötigen wir drei Zeiger, die sich unabhängig voneinander drehen und die Stunde, Minute und Sekunde anzeigen.

Wenn wir einen Zeiger auf einem Kreis anordnen, bedeutet dies, dass wir die Zeit in einen Winkel umwandeln müssen. Wenn wir also beispielsweise Mitternacht darstellen, haben alle Zeiger den Winkel 0. Wenn wir drei Uhr darstellen, hat der Minutenzeiger 0 Grad, der Stundenzeiger hat 90 Grad. Etwas schwieriger ist es, wenn wir den Minutenzeiger drehen. Stellen wir diesen beispielsweise auf 30 Minuten, dann hat dies auch Einfluss auf den Stundenzeiger. Bei halb vier Uhr wäre der Minutenzeiger 180 Grad (30 Minuten), der Stundenzeiger wäre 90 Grad + 1/24 des Kreises, d. h. 15 Grad, da wir die 30 Minuten ja noch berücksichtigen müssen.

Bevor wir uns in den Code stürzen, probieren wir einfach einmal das Programm aus und experimentieren mit dem Debugger.



5.2.1 Ausprobieren

1. Starte das Programm
Starte das Programm **«Analog watch»** und beobachte die Ausgabe.

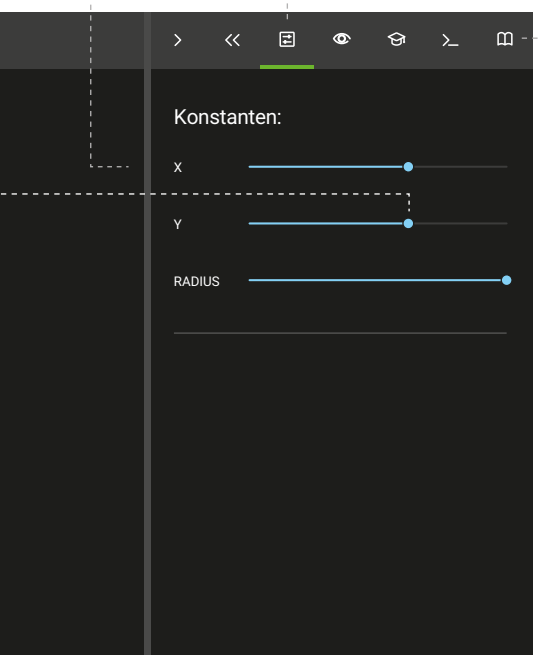
2. Experimentiere mit den Konstanten
Ändere die **X**- und **Y**-Koordinate ab. Dies verschiebt den Mittelpunkt der Uhr. Wo werden die Konstanten **«X»** und **«Y»** im Programm benutzt? Klicke hierzu **«X»** im Editor an, wodurch alle **«X»** hervorgehoben werden.

Ändere den Radius. Reduziere diesen um einen beliebigen Wert. Wo wird **«RADIUS»** überall im Programm verwendet?

3. Debugging
Starte den Debugger im Schritt-für-Schritt-Modus. Beobachte, was geschieht, nachdem die Zeile `update()` aufgerufen wird.

Lies den Eintrag zum Befehl `drawLine` in der **Online-Dokumentation*** und versuche zu verstehen, wie die Zeiger gezeichnet werden.

2. Experimentiere mit den Konstanten



***Online-Dokumentation**

X
Verschiebt die horizontale Position der Anzeige (0 = linker Rand, 239 = rechter Rand).

Y
Verschiebt die vertikale Position des Textes (0 = oberer Rand, 239 = unterer Rand).

RADIUS
Verändert den Radius der Uhr in Pixel.

5.2.2 Wie funktioniert es?

Die Herausforderung besteht darin, die Zahlen in Winkel zwischen 0° und 360° umzurechnen, sodass die entsprechenden Zeiger an der richtigen Stelle gezeichnet werden können. Hierzu verwenden wir die Grafikbefehle `translate` und `rotate`. Mit `translate` lässt sich der Nullpunkt verschieben, den wir in diesem Programm auf die Mitte des Bildschirms legen. Dies ist auch unsere neue Rotationsachse, um die wir die Zeiger im entsprechenden Winkel drehen werden.

Es gibt verschiedene Möglichkeiten, wie man eine analoge Uhr programmieren kann. Dank des `rotate`-Befehls finden wir hier eine elegantere Lösung, bei der nur etwas Algebra-Kenntnisse notwendig sind, um von der Minute / Stunde / Sekunde auf den Winkel zu kommen.

```
1  const X = 120            # 60..180
2  const Y = 120            # 60..180
3  const RADIUS = 110          # 40..110
4
5  def onDraw():
6      a:float
7
8      clear()
9      translate(X,Y)
10
11     stroke(255,255,255)
12     drawCircle(0,0,RADIUS)
13     update()
14
15
16     a = (360/60) * getMinute()
17     rotate(rad(a))
18     drawLine(0,0,0,-0.8*RADIUS)     # Paint the minutes
19     update()
20     rotate(-rad(a))
21
22     a = (360/12) * getHour()
23     a = a + (1.0/60.0)*getMinute() * (360/12)
24     rotate(rad(a))
25     stroke(150,150,150)
26     drawLine(0,0,0,-0.5*RADIUS)     # Paint the hour
27     update()
28     rotate(-rad(a))
29
30     a = (360/60)*getSecond()
31     rotate(rad(a))
32     stroke(255,0,0)
33     drawLine(0,0,0,-RADIUS+5)       # Paint the seconds
34     update()
35     rotate(-rad(a))
36
37     translate(-X,-Y)
38
39     delay(1000)
```

1-3 Hier finden wir unsere drei Konstanten, die wir im Konstanteneditor anpassen können.

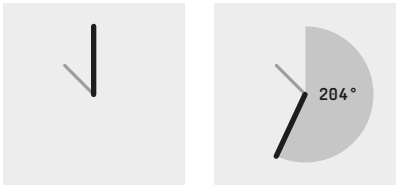
9 `translate` verschiebt den Nullpunkt des Bildschirms. Das ist der Punkt, bei dem das Pixel mit der Koordinate X = 0 und Y = 0 gezeichnet wird. Normalerweise ist der Nullpunkt oben links. Mit diesem Befehl kann man diesen jedoch verschieben. Für unsere analoge Uhr setzen wir den Nullpunkt auf die Mitte des Screens (120, 120) und auf **Zeile 37** machen wir es wieder rückgängig.

11 Wir bestimmen die Strichfarbe.

12 Zeichnet den Umkreis der Uhr. Als Radius verwenden wir die Konstante `RADIUS`.

16 Wir berechnen die Anzahl Grad, die wir den Zeiger im Uhrzeigersinn bewegen müssen, damit wir den Minutenzeiger positionieren können. Wenn wir die 360 durch 60 teilen, erhalten wir die Anzahl Grad für eine Minute. Multipliziert mit den Minuten erhalten wir die gesuchte Gradzahl.

11:34




```
360°/60 * getMinute()
6° * 34 =204°
```

17 Mit `rotate` rotieren wir nun die Achse um den neuen Nullpunkt. Die Funktion `rad` wandelt die Grad in Bogenmass um, da rotate Bogenmass erwartet.

18 Wir zeichnen nun mit `drawLine` eine Linie vom Mittelpunkt nach oben `(-y)` und verwenden hierzu die `RADIUS`-Konstante. Der Zeiger soll 0.8-mal der Radiuslänge entsprechen. Durch die zuvor formulierte `rotate`-Anweisung wird diese Linie vor dem Zeichnen noch rotiert.

20 Dies macht die zuvor umgesetzte Rotation wieder rückgängig.

22 Der Winkel des Stundenzeigers berechnet sich in zwei Schritten: Zuerst berechnen wir den Winkel in Bezug auf die aktuelle Stunde. Das geschieht in dieser Zeile.

23 In einem zweiten Schritt berechnen wir, wie weit der Stundenzeiger für jede vergangene Minute nach der vollen Stunde weiterwandert. Der Winkel zwischen zwei vollen Stunden beträgt je 30°, da 360°÷12=30°. Diesen Winkel durchläuft der Stundenzeiger innerhalb von 60 Minuten. Also rechnet man: 30°÷60=0.5°. Mit `0.5°*getMinute()` lässt sich ausrechnen, wie

weit der Stundenzeiger noch gewandert ist nach der vollen Stunde.

23-27 Wir rotieren wieder und zeichnen den nächsten Zeiger.

29-34 Als Letztes zeichnen wir den Sekundenzeiger nach demselben Prinzip.

37 Am Schluss setzen wir den Ursprung wieder zurück, damit beim nächsten `onDraw`-Aufruf alles korrekt gezeichnet werden kann.

5.2.3 Experimentieren

1. Gestaltung der Uhr verändern
Nutze folgende Befehle, um die Uhr nach den eigenen Vorstellungen zu gestalten.
- `stroke/strokeHSV`
- `fill/fillHSV`
- `drawLine/drawCircle`

2. Digital und analog
Kombiniere die beiden Beispiele und erstelle eine Uhr, die sowohl digitale als auch analoge Anzeigen kombiniert. Nutze `translate`, um die Zeichnungen zu verschieben.

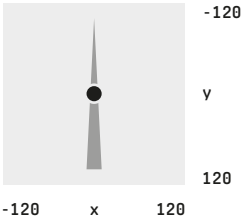
Wichtig: `translate` muss vor dem Verlassen des Zeichnungsbefehls wieder rückgängig gemacht werden, indem man translate nochmals mit denselben negierten Werten aufruft, d. h., aus `rotate`(a) wird `rotate`(-a), um die Rotation wieder rückgängig zu machen.

3. Dreieckige Zeiger
Bei analogen Uhren werden häufig dreieckige oder viereckige Zeiger verwendet. Die Funktionen hierzu heissen `drawTriangle` und `drawRectangle`. Wie müsste man das Programm anpassen, um beim Stundenzeiger ein Dreieck und Minutenzeiger ein Viereck zu zeichnen?

Wichtig: Der Drehpunkt des Dreiecks muss im Nullpunkt liegen, da nur um den Nullpunkt rotiert werden kann.

Beispiel:

```
translate(120, 120)
drawTriangle(0,-50,-5,50,5,50)
```



4. Millisekunden anzeigen
Die Funktion `millis` liefert die Anzahl Millisekunden, die seit dem Start der Oxocard verstrichen sind. Durch die folgende Formel erhalten wir die verstrichenen Millisekunden der aktuellen Zeit. (% ist die Modulo-Funktion – weitere Informationen finden sich in der Anleitung.)

```
millis() % 1000
```

Versuch nun, mit dieser Formel einen weiteren Block zu programmieren, der neben den Stunden, Minuten und Sekunden auch noch die Millisekunden als Kreisbruchteil darstellt.

Hinweis: Die Rotation einer Millisekunde entspricht einem 1/1000 des Kreisumfangs: 360°÷1000=0.36°.

5.2.4 Weiterführende Informationen

6.6.2 Deklarieren von Variablen
6.6.3 Rechnen mit Variablen
6.8 Rechnen mit eingebauten Funktionen
8.1.11 deg(rad)
8.3.7 Transformieren, rotieren und skalieren

5.2.5 Wichtig zu wissen

Variablen sind Platzhalter für Zahlen, mit denen man – wie in der Algebra gelernt – rechnen kann.

Neben Zahlen und anderen Variablen kann man in Berechnungen auch Funktionen verwenden, beispielsweise `getHour`, `getMinute` und `getSecond`.
Auf dem Oxocard-Screen lässt sich mit `translate` der Nullpunkt verschieben. Diese Verschiebung bleibt bestehen, man kann sie jedoch rückgängig machen, indem man `translate` nochmals mit den negativen Werten aufruft.

Mit `rotate` kann man die Bildschirmausgabe im Uhrzeigersinn (positive Zahl) oder Gegenuhrzeigersinn (negative Zahl) drehen. Die Funktion erwartet einen Wert im Bogenmass (0 bis 2π). Mit der rad-Funktion kann man Grad in Bogenmass umrechnen.

5.3 Zeit: Wort-Uhr

In unserem dritten Uhrenbeispiel wandeln wir die Ziffern in Worte um. Wir schreiben «fünf nach halb drei» für 14:35 Uhr und schreiben «bald fünf nach halb drei», wenn die Zeit zwischen 14:31 und 14:35 liegt.

Dies gestaltet sich als aufwendiger, als es auf den ersten Blick scheint. Wir möchten, dass uns die Oxocard die Zeit als «halb zehn» und nicht als «neun Uhr dreissig» ansagt. Bevor wir mit der Entwicklung eines Algorithmus starten, müssen wir uns zuerst gut überlegen, welche Regeln gelten, wenn wir die Zeit sagen.

Interessant ist beispielweise, dass wir «zwanzig nach neun» sagen, fünf Minuten später wechselt es aber zu «fünf vor halb zehn». Und eine Viertelstunde später sagen wir wieder «zwanzig vor zehn». Alle diese Aspekte möchten wir berücksichtigen.

SET_HOUR
Fixe Stunde (siehe INTERNET_TIME)

SET_MINUTE
Fixe Minute (siehe INTERNET_TIME)

INTERNET_TIME
Wenn dieser Schalter **«ein»** ist, wird die aktuelle Zeit angezeigt, wenn dieser Schalter **«aus»** ist, kann man mit **SET_HOUR/SET_MINUTE** eine bestimmte Zeit einstellen. Damit lässt sich das Programm zu verschiedenen Zeiten analysieren.

3. Debugging I

```
1   const SET_HOUR = 12          # 0..23
2   const SET_HOUR = 12          # 0..23
3   const INTERNET_TIME = true   # true..false
4
5   def hourToText(hour)->byte[20]:
6       text:byte[20]
7       if hour==0 or hour==12: text = "zwölf"
8       if hour==1:             text = "eins"
9       elif hour==2:           text = "zwei"
10      elif hour==3:           text = "drei"
11      elif hour==4:           text = "vier"
```

Konstanten:

SET_HOUR

SET_MINUTE

INTERNET_TIME

5.3.1 Ausprobieren

1. Starte das Programm
Starte das Programm **«Text watch»** und beobachte die Ausgabe.

2. Experimentiere mit den Konstanten
Setze `INTERNET_TIME` auf **«aus»** und experimentiere mit verschiedenen Zeiten, die mit `SET_HOUR` und `SET_MINUTE` hinterlegt werden können.

3. Debugging I
Wir untersuchen den Programmablauf.

1. Setze den Schalter `INTERNET_TIME` auf **«ein»** und lass das Programm durchspielen. Beobachte, was auf den Zeilen **55** und **57** geschieht

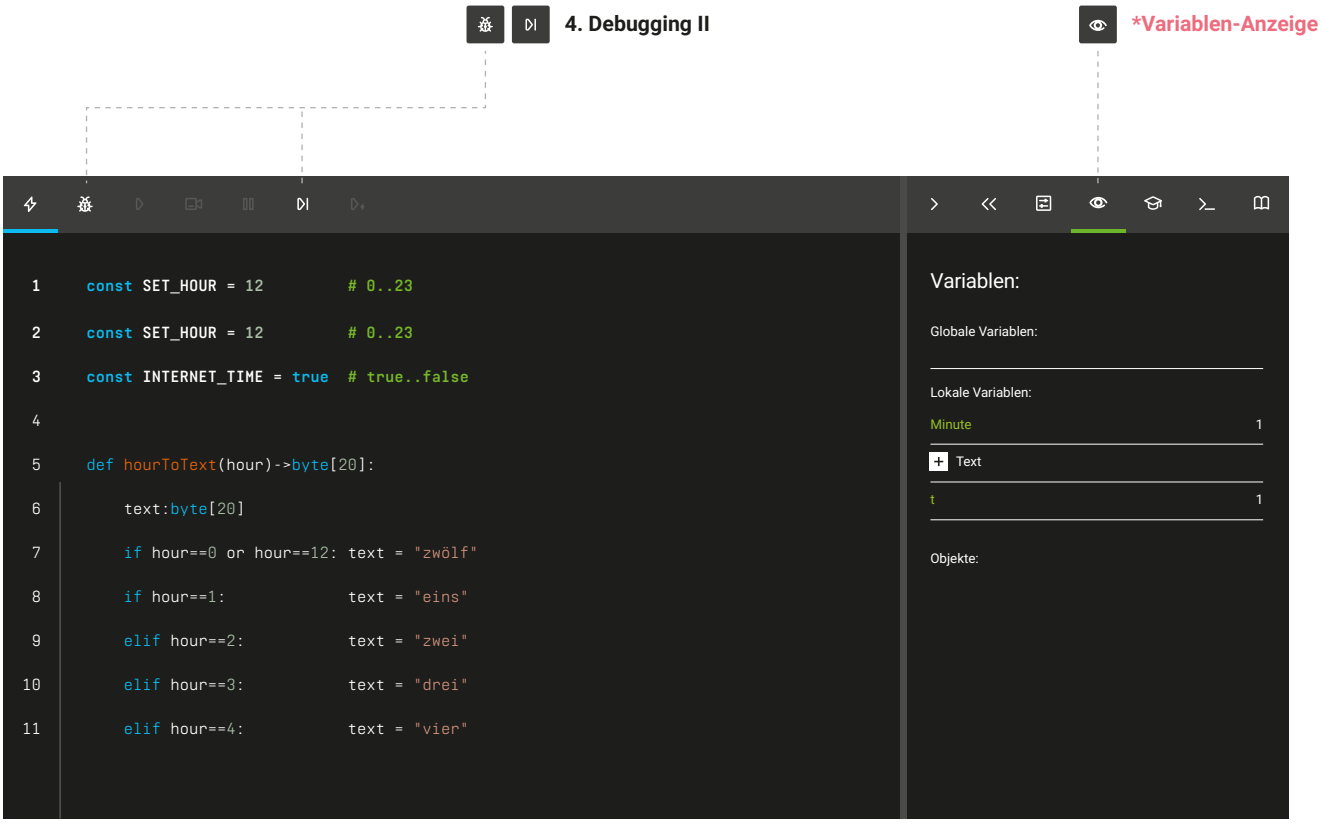
2. Setze den Schalter `INTERNET_TIME` auf **«aus»**, setze die Konstanten `SET_HOUR` und `SET_MINUTE` auf folgende Zeiten: 09:20, 09:25, 12:01, 12:05, 14:45, 2:45 – lass den Code mit dem Debugger Schritt für Schritt laufen und untersuche, welche Zeilen ausgeführt werden:

Welche Unterschiede stellt du zwischen 09:20 und 09:25 fest?
Welche Unterschiede gibt es zwischen 12:01 und 12:05?
Welche Unterschiede gibt es zwischen 14:45 und 2:45?

4. Debugging II
Starte den Debugger im Schritt-für-Schritt-Modus.
Wechsle in der rechten Bildschirmseite auf die Variablen-Anzeige*.

Starte das Programm jeweils mit folgenden Zeiten:
09:15, 09:25, 09:30, 09:45, 10:00

Untersuche im Variablen-Fenster, wie sich die Variablen `hour`, `minute` und `t` verändern.



5.3.2 Wie funktioniert es?

Unser Code besteht aus zwei Funktionen, die jeweils die Stunden und die Minuten in Text umwandeln.
Bei den Stunden wandeln wir das 24h-Format zuerst in das 12h-Format und geben für jede Zahl den entsprechenden Text aus. Die Minuten unterteilen wir in Fünfminutenblöcke und schreiben uns zu jedem Block den entsprechenden Text. Jetzt gibt es noch zwei Spezialfälle: Wenn die Zeit nicht exakt auf fünf Minuten fällt, erhöhen wir auf die nächsten fünf Minuten und schreiben **"bald"**, wie wir das umgangssprachlich auch machen. Wenn die Minuten > 25 sind, erhöhen wir die Stunde.

In der `onDraw`-Funktion wird alles zusammengesetzt.

```
1  const SET_HOUR = 12          # 0..23
2  const SET_MINUTE = 30        # 0..59
3  const INTERNET_TIME = true   # true..false
4
5  def hourToText(hour)->byte[20]:
6      text:byte[20]
7      if hour==0 or hour==12: text = "zwölf"
8      if hour==1:             text = "eins"
9      elif hour==2:           text = "zwei"
10     elif hour==3:           text = "drei"
11     elif hour==4:           text = "vier"
12     elif hour==5:           text = "fünf"
13     elif hour==6:           text = "sechs"
14     elif hour==7:           text = "sieben"
15     elif hour==8:           text = "acht"
16     elif hour==9:           text = "neun"
17     elif hour==10:          text = "zehn"
18     elif hour==11:          text = "elf"
19     return text
20
21 def minuteToText(minute)->byte[30]:
22     text:byte[30]
23     t = minute / 5
24     if minute%5 != 0:
25         t = (t+1) % 12
26     if t == 1:      text = "fünf nach"      # 5
27     elif t == 2:    text = "zehn nach"      # 10
28     elif t == 3:    text = "viertel nach"   # 15
29     elif t == 4:    text = "zwanzig nach"   # 20
30     elif t == 5:    text = "fünf vor halb"  # 25
31     elif t == 6:    text = "halb"           # 30
32     elif t == 7:    text = "fünf nach halb" # 35
33     elif t == 8:    text = "zwanzig vor"    # 40
34     elif t == 9:    text = "viertel vor"    # 45
35     elif t == 10:   text = "zehn vor"       # 50
36     elif t == 11:   text = "fünf vor"       # 55
37     if minute%5 != 0:
38         text = "bald " + text
39     return text
40
41 def onDraw():
42     if INTERNET_TIME:
43         h = getHour()
44         minute = getMinute()
45     else:
46         h = SET_HOUR
47         minute = SET_MINUTE
48
49     clear()
50     hour = h % 12
51     if minute >=25:
```

5-19 Wir definieren hier eine Funktion, mit der die Stunde in einen Text umgewandelt werden kann. Die Funktion ist nach dem EVA-Prinzip aufgebaut: Eingabe – Verarbeitung – Ausgabe. Als Eingabe übergeben wir eine Zahl, die der Stunde entspricht. Die Verarbeitung wandelt diese Zahl in einen Text um und liefert als Ausgabe den entsprechenden Text.
Die Zeilen sind jeweils einfach lesbar. Auf Zeile **7** lesen wir, dass **«zwölf»** zurückgegeben werden soll, wenn die Stunde 0 oder 12 ist.

21-39 Diese Funktion macht dasselbe für die Minuten. Sie teilt die Minuten jeweils in Fünfminutenblöcke, die dann textlich konvertiert werden. Spezialfall: Wenn die Zeit nicht exakt auf einen der Fünfminutenblöcke fällt, wird noch **«bald»** geschrieben.

55 Hier wird die zuvor definierte Funktion `minuteToText` verwendet. In Klammern steht als Input die aktuelle Minutenzahl. Wie in der Funktion definiert, wird ein Text zurückgegeben, der dann mit `drawText` ausgegeben wird.

23 Wir betrachten die Funktion `minuteToText` etwas genauer: Auf dieser Zeile teilen wir die Anzahl Minuten durch 5 und speichern diese in der Variablen t. Diese Variable enthält also eine Zahl zwischen 0 und 12.
Bei 0 und 12 müssen wir nichts ausgeben.
In allen anderen Fällen haben wir wieder Texte definiert.

24-25 Diese Zeilen decken einen Spezialfall ab:

37-38 Wenn die Minuten nicht exakt auf eine durch 5 restlos teilbare Zahl fallen, z. B. 23 oder 46, schreiben wir noch **"bald"**. Das %-Zeichen ist eine neue Rechnungsfunktion, die den Restwert einer ganzzahligen Division liefert und Modulo heisst. 23%5 = 3, 46%5 = 1. Wir sagen dann 23 modulo 5 ist 3 oder 46 modulo 5 ist 1. Wenn es also einen Rest gibt, erhöhen wir auf die nächsten 5 Minuten (Zeilen 24/25) und schreiben **«bald»** (Zeile 37/38).

50 Wir nutzen hier die Modulo-Funktion, um die Stunden im 24-Stundenformat in das 12-Stundenformat umzuwandeln. Wenn wir z. B. 23 Uhr durch 12 teilen und uns den Rest anschauen, erhalten wir 11, da 23%12 = 11.

51-52 Wenn die aktuelle Minute grösser als 25 ist, sagen wir ja nicht mehr 11 Uhr 25, sondern fünf vor halb zwölf. Daher erhöhen wir in dem Fall die Stunden um eins.

```
52          hour = (hour+1) % 12
53
54      textFont(FONT_ROBOTO_24)
55      drawText(10,80,minuteToText(minute))
56      textFont(FONT_ROBOTO_48)
57      drawText(10,110,hourToText(hour))
58      update()
59      delay(1000)
```

5.3.3 Experimentieren

1. Gestaltung der Uhr verändern
Wir gestalten als Erstes die Uhr wieder nach unseren Wünschen um.

Neben den bereits bekannten Zeichnungsbefehlen lassen sich mit dem neuen `textFont` die Schriftart und -grösse bestimmen. Die genaue Funktionsweise ist der Online-Hilfe zu entnehmen.

2. Übersetzung in andere Sprachen
Übersetze die Texte in Mundart, Französisch, Englisch oder eine beliebige andere Sprache. Prüfe, ob die verwendeten Algorithmen auch in der anderen Sprache funktionieren oder ob gegebenenfalls noch Anpassungen notwendig sind.

3. Alarm-Funktion
Ergänze folgende zwei Konstanten:

```
const ALARM_HOUR = 0 # 0..23
const ALARM_MINUTE = 0 # 0..59
```

Bau das Programm nun so aus, dass vor der Zeit in Rot **«ALARM»** steht, wenn die aktuelle Zeit der eingestellten Alarm-Zeit entspricht.

5.3.4 Weiterführende Informationen

6.6.3 Rechnen mit Variablen
6.11.3 Funktionen mit Rückgabewerten
6.14.2 Texte (Strings)

5.3.5 Wichtig zu wissen

Mit Funktionen kann man Code strukturieren, indem man einen Block von Anweisungen mit einem Namen versieht.

Funktionen können auch Daten zurückgeben, wobei man hierzu im Funktionskopf definieren muss, welche Form die Daten haben.

Es gibt auch Textvariablen, die einen beliebigen Text speichern können. Im Computerspeicher werden diese als Liste von Bytes gespeichert, wobei ein Byte eine Zahl zwischen 0 und 255 speichern kann. Textvariablen werden mit `variable:byte [länge]` deklariert, also z. B. `text:byte[100]`.

Die **Modulo-Funktion %** liefert den Restwert einer ganzzahligen Division.

Mit `textFont` lassen sich die Schriftgrösse und Schriftgrad bestimmen.

Damit du nichts verpasst, lohnt es sich, einen der folgenden Kanäle regelmässig zu besuchen.

www.oxocard.ch

Auf der Homepage werden neue Produkte präsentiert. Zudem informieren wir hier auch über Veranstaltungen und Weiterbildungskurse.

Instagram

Möchtest du dich durch andere Beispiele inspirieren lassen? Dann besuch uns auf Instagram. Hier werden laufend coole Programme publiziert – häufig auch mit Quellangaben zum Code.

Youtube

Auf Youtube findest du Informationsvideos und Tutorials. Die Liste wird laufend ergänzt.

Discord

Discord ist unsere Chat- und Hilfeplattform. Melde dich jetzt an und chatte direkt mit den Entwickler*innen der Oxocard.