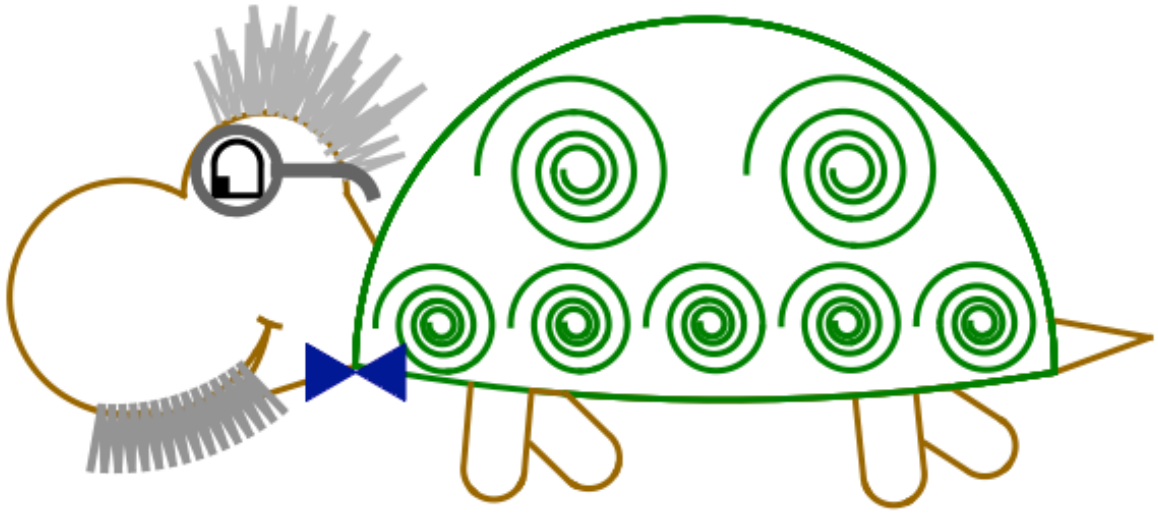
ABZ AUSBILDUNGS- UND BERATUNGSZENTRUM
FÜR INFORMATIKUNTERRICHT

Hans-Joachim Böckenhauer Juraj Hromkovič Dennis Komm

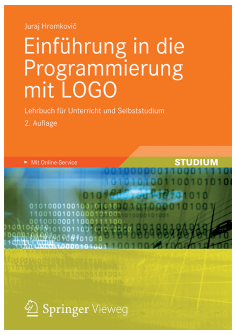
Programmieren mit LOGO
Projekte



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programmieren mit LOGO

Dieses Skript baut auf dem Lehrbuch *Einführung in die Programmierung mit LOGO* auf. Das Lehrbuch enthält viele weitere Aufgaben und Erklärungen. Ausserdem ist es mit Hinweisen für die Lehrperson versehen. Das Lehrbuch umfasst insgesamt 15 Lektionen.



Juraj Hromkovič. *Einführung in die Programmierung mit LOGO: Lehrbuch für Unterricht und Selbststudium*. 2. Aufl., Springer Vieweg 2012. ISBN: 978-3-8348-1852-2.

Version 0.35, 22. Februar 2016, SVN-Rev: 17661

Programmierumgebung

Die vorliegenden Unterrichtsunterlagen wurden für die Programmierumgebung XLogo entwickelt. XLogo ist auf der Webseite `xlogo.tuxfamily.org` kostenlos verfügbar.

Damit die Logo-Programme aus den Unterlagen ausgeführt werden können, muss XLogo auf Englisch eingestellt werden.

Nutzungsrechte

ABZ

Das Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich unterstützt Schulen und Lehrkräfte, die ihren Informatikunterricht entsprechend auf- oder ausbauen möchten, mit einem vielfältigen Angebot. Es reicht von individueller Beratung und Unterricht durch ETH-Professoren und das ABZ-Team direkt vor Ort in den Schulen über Ausbildungs- und Weiterbildungskurse für Lehrkräfte bis zu Unterrichtsmaterialien.

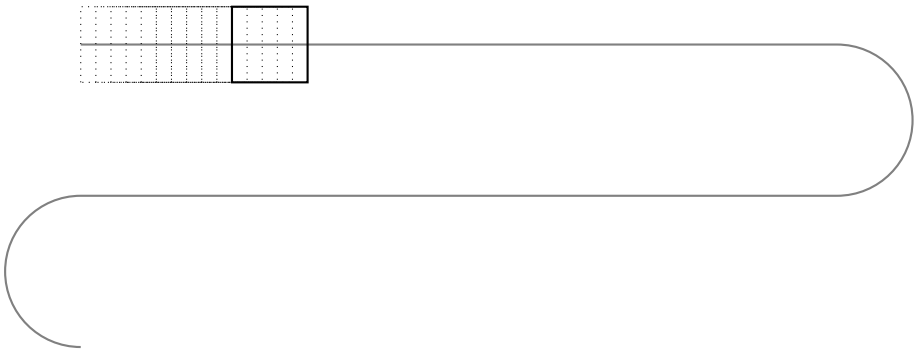
`www.abz.inf.ethz.ch`

Voraussetzung

Zur Bearbeitung der folgenden Projekte sollten zunächst die beiden Skripte „Programmieren mit LOGO“ und „Programmieren mit LOGO für Fortgeschrittene“ bearbeitet worden sein. Das in diesen Unterlagen vermittelte Wissen wird im Folgenden vorausgesetzt.

Projekt 1. *Animationen*

In den Kapiteln 7 und 8 haben wir uns mit Animationen beschäftigt. Nun wollen wir dieses Thema aufgreifen und ein Quadrat in Schlangenlinien von oben nach unten auf dem Bildschirm laufen lassen. Schematisch ist dies im folgenden Bild dargestellt.



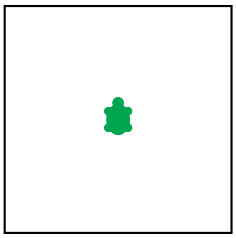
Wir gehen schrittweise vor, um die Animation zu erstellen.

1. Das zu zeichnende Grundobjekt erstellen.
2. Das Objekt auf einer Geraden fester Länge bewegen.
3. Parameter hinzufügen.
4. Das Objekt eine Rechtskurve beschreiben lassen.
5. Das Objekt analog eine Linkskurve beschreiben lassen.
6. Das Objekt Schlangenlinien beschreiben lassen.

Beginnen wir mit dem Zeichnen des zu bewegenden Objektes.

Schritt 1

Falls du ein solches Programm nicht schon erstellt hast, erstelle ein Programm `QUADMITTE`, das ein Quadrat aus der Mitte heraus zeichnet. Die Seitenlänge ist hierbei wieder mit einem Parameter `:GR` frei wählbar. Wichtig ist, dass die Schildkröte vor und nach dem Zeichnen des Quadrates in dessen Mitte steht.



Jetzt können wir das Quadrat zunächst von links nach rechts über den Bildschirm bewegen. Wieder verwenden wir zum Überzeichnen mit dem weissen Stift einen grösseren Stift-Durchmesser, damit das Quadrat keine „Spur“ hinterlässt.

```
to GEHERECHTS
ht
setpw 3
ppt
QUADMITTE 100
wait 2
repeat 100 [
  setpw 10
  pe
  QUADMITTE 100
  pu rt 90 fd 1 lt 90 pd
  setpw 3
  ppt
  QUADMITTE 100
  wait 2
]
end
```

Das Quadrat wird also zunächst auf der Ausgangsposition gezeichnet. Dann warten wir kurz und die Animation beginnt. Am Ende befindet sich das Quadrat genau 100 Positionen weiter rechts als zu Beginn.

Schritt 2

Teste das Programm `GEHERECHTS` und vollziehe die einzelnen Zeilen genau nach.

Dieses Programm soll als nächstes flexibler gemacht werden.

Schritt 3

Füge `GEHERECHTS` drei Parameter hinzu, und zwar `:GR` für die Seitenlänge des zu zeichnenden Quadrats, `:PAUSE` für die Anzahl der Zeiteinheiten, die zwischen den einzelnen Bildern gewartet werden soll, und `:WDH` für die Anzahl der Wiederholungen.

Teste das Programm mit verschiedenen Parameterwerten.

Wir erinnern uns, dass wir einen Halbkreis beispielsweise mit

```
repeat 180 [fd 1 rt 1]
```

zeichnen können. Wenn wir die Schildkröte vorher um 90° nach rechts drehen, so erhalten wir das folgende Bild.

Mit derselben Idee können wir auch unser Quadrat einen Halbkreis beschreiben lassen.

```
to KURVERECHTS
ht
rt 90
setpw 3
ppt
QUADMITTE 100
wait 2
repeat 180 [
  setpw 10
  pe
  QUADMITTE 100
  pu fd 1 rt 1 pd
  setpw 3
  ppt
  QUADMITTE 100
  wait 2
]
lt 90
end
```

Analog zu `GEHERECHTS` ist hierbei zu beachten, dass wir das Quadrat nicht 180-mal zeichnen, sondern 181-mal. Zunächst wird es auf der Ausgangsposition gezeichnet, also um 0° gedreht. Die folgenden 180 Wiederholungen drehen es dann 180-mal, bis das Quadrat einen Halbkreis beschrieben hat. Zu Beginn drehen wir uns um 90° nach rechts, um das erwünschte Bild zu zeichnen. Am Ende drehen wir uns wieder zurück.

Schritt 4

Füge `KURVERECHTS` zwei Parameter hinzu, und zwar `:UM` für den Umfang des zu zeichnenden Kreises und `:PAUSE` für die Anzahl der Zeiteinheiten, die zwischen den einzelnen Bildern gewartet werden soll.

Teste das Programm mit verschiedenen Parameterwerten.

Hinweis: Wir erinnern uns, dass die Länge, in die sich die Schildkröte in jedem Zeitschritt bewegt, dem Umfang geteilt durch 180 entspricht. Speichere diesen Wert zu Beginn in einer Variablen, die du dann verwendest.

Schritt 5

Erstelle ein Programm `KURVELINKS`, das analog zu `KURVERECHTS` funktioniert.

Jetzt haben wir alle Zutaten für den letzten Schritt.

Schritt 6 (Schlangenlinien)

Nun wollen wir ein Programm `SCHLANGE` erstellen, das ein Quadrat mit angegebener Seitengrösse in Schlangenlinien über den Bildschirm von oben nach unten bewegt. Hierbei sollen die oben erstellten Programme als Unterprogramme verwendet werden.

Ein „Abschnitt“ der Schlangenlinie besteht aus der Abfolge der erstellten Programme `GEHERECHTS`, `KURVERECHTS` und `KURVELINKS`.

`SCHLANGE` erhält insgesamt fünf Parameter:

1. `:ABSCH`, mit dem die Anzahl dieser Abschnitte angegeben werden kann.
2. `:GR`, der die Grösse des Quadrates angibt und dessen Wert an die entsprechenden Unterprogramme weitergegeben wird.
3. `:PAUSE`, dessen Wert als Wartezeit an die Unterprogramme übergeben wird.
4. `:LAENGE` gibt an, wie lang die geraden Strecken sein sollen, die mit `GEHERECHTS` jeweils zurück gelegt werden.
5. `:UM` bestimmt den Umfang der Halbkreise, die mit `KURVERECHTS` und `KURVELINKS` gezeichnet werden.

Eine Ausführung von `SCHLANGE 1 40 1 200 80` soll beispielsweise zu der am Anfang dieses Kapitels dargestellten Animation führen.

Wir können an dieser Stelle unser Programm mit diversen Ideen erweitern. Die Bearbeitung der folgenden Punkte ist freiwillig.

- Erstelle ein Programm `KURVE`, das einen weiteren Parameter `:RICHTUNG` erhält und eine Linkskurve macht, wenn der Wert dieses Parameters 1 ist und eine Rechtskurve macht, wenn er 2 ist.
- Bewege andere Objekte als ein Quadrat.
- Lasse die Objekte komplexere Bahnen beschreiben.

Projekt 2. *Türme auf einem Schachbrett*

In diesem Kapitel wollen wir Türme auf einem Schachbrett positionieren. Dies soll so geschehen, dass pro Zeile und Spalte nur jeweils ein Turm steht, sodass sich keine zwei Türme bedrohen (wie wir wissen, können Türme sich nur horizontal und vertikal bewegen). Wir gehen wieder in Schritten vor.

1. Das Schachbrett zeichnen, sodass die Anzahl der Felder mit Parametern angegeben werden kann.
2. Farbige Rechtecke zeichnen, aus denen wir dann Türme zusammensetzen können.
3. Türme auf dem Schachbrett platzieren.
4. 8 Türme so auf dem Schachbrett platzieren, dass sie sich nicht bedrohen.

In Kapitel 3 wurden die Programme `FETT` und `SCHWARZ40` besprochen, mit denen ausgefüllte Quadrate gezeichnet werden können. Wir verwenden diese Technik im folgenden Programm, um ein ausgefülltes Quadrat der Seitenlänge `:GR` darzustellen.

```
to GEFQUAD :GR
repeat :GR [fd :GR rt 90 fd 1 rt 90 fd :GR rt 180]
end
```

Nun können wir mit dem folgenden Programm ein Schachbrett zeichnen.* Mit den Parametern `:NUM` und `:GR` geben wir an, wie viele Quadrate wir pro Reihe und Spalte zeichnen wollen und wie gross die einzelnen Quadrate sein sollen.

```
to SCHACHBRETT :NUM :GR
make "REST mod :NUM 2
if :REST = 1 [
  pr [Anzahl von Quadraten pro Reihe muss gerade sein.] stopall
] []
cs ht
repeat :NUM/2 [
  repeat :NUM/2 [setpc 0 GEFQUAD :GR setpc 7 GEFQUAD :GR]
  pu lt 90 fd :NUM*:GR rt 90 fd :GR pd
  repeat :NUM/2 [setpc 7 GEFQUAD :GR setpc 0 GEFQUAD :GR]
  pu lt 90 fd :NUM*:GR rt 90 fd :GR pd
]
rt 90 repeat 4 [fd :NUM*:GR rt 90]
lt 90 bk :NUM*:GR
end
```

*Eigentlich ist dieses Schachbrett nicht ganz symmetrisch, aber wir ziehen es vor, hier eine möglichst einfache Darstellung zu wählen.

Die Schildkröte steht nach dem Zeichnen des Spielfelds an dessen Ecke unten links. Wichtig ist ausserdem, dass wir zu Beginn prüfen, ob die Anzahl der Quadrate pro Reihe (und somit Spalte) gerade ist, um zu garantieren, dass ein vernünftiges Schachbrett gezeichnet wird. Da wir SCHACHBRETT später als Unterprogramm verwenden möchten, benutzen wir hier den Befehl **stopall**, sodass auch das entsprechende Hauptprogramm beendet wird, wenn SCHACHBRETT mit unzulässigen Parametern aufgerufen wird.

Schritt 1

Teste das Programm SCHACHBRETT zunächst mit verschiedenen Werten.

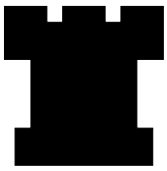
Als nächstes wollen wir Türme auf dem Schachbrett platzieren. Hierzu erstellen wir ein Programm, das farbige Rechtecke darstellt, aus denen wir dann die Türme zusammensetzen, die wir anschliessend auf das Schachbrett setzen.

Schritt 2

Schreibe ein Programm GEFRECHTECK mit drei Parametern :HOR, :VER und :FARBE. Analog zu GEFQUAD soll dieses Programm ein ausgefülltes Rechteck zeichnen, dessen Höhe und Breite mit den ersten beiden Parametern angegeben werden und das die Farbe besitzt, die der Wert des dritten Parameters repräsentiert.

Mit dem folgenden Programm können wir unter Verwendung von GEFRECHTECK einen Turm zeichnen (wie schematisch dargestellt).

```
to TURM :FARBE
GEFRECHTECK 30 6 :FARBE
pu lt 90 fd 25 rt 90 fd 6 pd
GEFRECHTECK 20 15 :FARBE
pu fd 15 lt 90 fd 27 rt 90 pd
repeat 2 [GEFRECHTECK 8 8 :FARBE GEFRECHTECK 5 5 :FARBE]
GEFRECHTECK 8 8 :FARBE
end
```



Mit dem nächsten Programm TURMAUF können wir einen Turm auf bestimmte Positionen des Schachbretts zeichnen, wenn dieses eine bestimmte Grösse besitzt. Anschliessend wird die Schildkröte wieder auf die Ausgangsposition gestellt. Dies passiert mit dem Befehl **home**. Anders als bei **cs** wird dabei der Bildschirm nicht gelöscht.

```
to TURMAUF :X :Y :FARBE
pu fd 40*(:Y-1)+5 rt 90 fd 40*(:X-1)+4 lt 90 pd
TURM :FARBE
pu home pd
end
```

Schritt 3

Tippe zunächst `SCHACHBRETT 8 40` ein und platziere dann Türme auf dem enstandenen Schachbrett mit `TURMAUF 1 1 1`, `TURMAUF 3 5 4` und `TURMAUF 6 2 1`. Die Farben Weiss und Schwarz wären in unserem Fall eine schlechte Wahl für die beiden Spieler, deswegen benutzen wir Rot (Farbe 1) und Blau (Farbe 4).

Nun können wir die Türme in der besprochenen Weise auf das Schachbrett stellen.

Schritt 4 (Türme auf dem Schachbrett)

Jetzt wollen wir also 8 Türme auf einem Schachbrett der Grösse 8×8 platzieren, sodass kein Turm einen anderen bedroht, das heisst, im nächsten Zug schlagen kann. Die Position eines dieser Türme ist dabei vorgegeben.

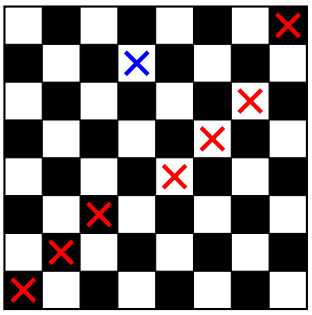
Schreibe hierzu ein Programm `TUERME`, das als erstes ein Schachbrett zeichnet, indem es `SCHACHBRETT` als Unterprogramm verwendet. Ferner hat `TUERME` zwei Parameter `:TURMX` und `:TURMY`, die einen blauen Turm auf die entsprechende Position des Schachbretts setzen.

```
to TUERME :TURMX :TURMY
if :TURMX>8 [print [Horizontale Position zu gross.] stop] []
if :TURMX<1 [print [Horizontale Position zu klein.] stop] []
if :TURMY>8 [print [Vertikale Position zu gross.] stop] []
if :TURMY<1 [print [Vertikale Position zu klein.] stop] []
SCHACHBRETT 8 40
TURMAUF :TURMX :TURMY 4
...
end
```

Danach sollen 7 rote Türme gezeichnet werden, die weder den gegebenen blauen Turm noch sich gegenseitig bedrohen.

Wenn wir insgesamt 8 Türme auf dem Spielfeld haben wollen, muss jeder in einer eigenen Spalte stehen. Der Turm in der ersten Spalte kann in 7 möglichen Zeilen stehen, denn in einer bedroht er den blauen Turm. Der zweite rote Turm kann dann noch in 6 möglichen Zeilen stehen und wir sehen schnell, dass es 7!, also 5040 Möglichkeiten gibt, die 7 roten Türme zu platzieren.

Eine mögliche Lösung könnte so vorgehen, dass alle roten Türme auf einer Diagonalen stehen (dann bedrohen sie sich nicht gegenseitig). Diese beginnt unten links, also auf Position (1,1), und endet oben rechts. Natürlich ist der blaue Turm nicht zwingend auf dieser Diagonalen platziert und so müssen wir seiner Position beim Platzieren der roten Türme „ausweichen“ wie rechts im Bild dargestellt.



Diese Idee können wir wie folgt umsetzen.

1. Definiere zwei Variablen `:X` und `:Y`, die zunächst beide den Wert 1 haben.
2. Wenn `:X` gleich `:TURMX` ist, erhöhe `:X` um 1.
3. Wenn `:Y` gleich `:TURMY` ist, erhöhe `:Y` um 1.
4. Zeichne einen roten Turm an die Position (`:X`,`:Y`) des Schachbretts.
5. Erhöhe `:X` und `:Y` um 1.
6. Wiederhole von 2. an bis insgesamt 8 Türme auf dem Schachbrett stehen.

Projekt 3. *Primzahlen*

In vielen Anwendungen der Informatik spielen Primzahlen, also natürliche Zahlen, die nur durch 1 und sich selbst teilbar und nicht selber 0 oder 1 sind, eine wichtige Rolle, insbesondere bei der Verschlüsselung von Daten. Wir wollen uns in diesem Projekt mit der Frage beschäftigen, wie viele Primzahlen einer bestimmten Grösse es gibt. Wir gehen wie folgt vor.

1. Testen, ob eine gegebene Zahl eine Primzahl ist.
2. Den Test verfeinern.
3. Den Test effizienter gestalten.
4. Den Test verwenden, um zu testen, ob eine gegebene Zahl eine Primzahlpotenz ist.

Das folgende Programm `PRIM` überprüft für einen gegebenen Parameter `:NUM`, ob dessen Wert einer Zahl entspricht, die eine Primzahl ist. Dies passiert in gleicher Weise, wie das Programm `GERADEZAHL` gearbeitet hat. Da alle natürlichen Zahlen durch 1 teilbar sind, muss nur überprüft werden, ob es eine weitere Zahl gibt, die den Wert von `:NUM` teilt und dabei kleiner ist als dieser. Falls der Wert von `:NUM` einer Primzahl entspricht, wird ein rotes Quadrat (Farbe 1) gezeichnet, sonst ein schwarzes (Farbe 2).

```
to PRIM :NUM
ht
make "IT 2
make "ISTPRIM 1
while [:IT<:NUM] [
  make "REST mod :NUM :IT
  if :REST=0 [make "ISTPRIM 0] []
  make "IT :IT+1
]
if :ISTPRIM=1 [setpc 1 QUADRAT 8]
              [setpc 0 QUADRAT 8]
end
```

Schritt 1

Vollziehe das Programm `PRIM` zunächst genau nach. Eine Schlüsselrolle spielt die Variable `:PRIM`, die speichert, ob der Wert von `:NUM` eine Primzahl ist oder nicht.

Überprüfe dann für verschiedene Werte für `:NUM`, ob es funktioniert.

Ergänze die folgende Tabelle für die angegebenen Ausführungen der **`while`**-Schleife, wenn `PRIM 7` ausgeführt wird. Hierbei entspricht der 0-te Zeitschritt dem Zeitschritt vor ihrer ersten Ausführung.

	0	1	2	3	4	5
`IT(PRIM)`	2	2	3	4	5	6
`ISTPRIM(PRIM)`	1	1				
`REST(PRIM)`	–	1				

`PRIM` arbeitet noch nicht für alle Eingaben korrekt, weil der Wert von `:IT` zu Beginn auf 2 gesetzt wird. Dies wollen wir jetzt ändern.

Schritt 2

Wie wir wissen, ist die Zahl 1 per Definition keine Primzahl. Der Aufruf von `PRIM 1` führt allerdings zu einer falschen Ausgabe.

Erweitere das Programm, indem du einen weiteren **`if`**-Befehl benutzt, sodass für 1 ein schwarzes Quadrat gezeichnet wird. Sorge ausserdem dafür, dass mit dem Befehl **`pr`** eine Fehlermeldung ausgegeben wird, wenn 0 oder eine negative Zahl übergeben werden. Verwende hierfür den Befehl **`stopall`**.

Nun können wir das Programm `PRIM` „schneller“ (effizienter) machen, indem wir die **`while`**-Schleife weniger oft durchlaufen lassen. Hierzu verwenden wir die folgende Idee.

Schritt 3

Nehmen wir an, eine Zahl x sei keine Primzahl. Dann existiert also eine Zahl a, die nicht 1 und nicht x ist und die x ohne Rest teilt. Daraus folgt aber wiederum, dass eine weitere Zahl b existieren muss, die x ebenfalls ohne Rest teilt und die weder 1 noch x ist.

Ein wichtiger Punkt ist, dass eine dieser beiden Zahlen höchstens so gross wie die Quadratwurzel von x ist. Falls beispielsweise a etwas grösser ist als $\sqrt{x}$, so muss b kleiner sein als $\sqrt{x}$, da $a \cdot b$ sonst grösser wäre als x.

Diese Beobachtung wollen wir jetzt ausnutzen, um das Programm `PRIM` zu verbessern. Schreibe ein Programm `PRIM2`, das genau wie `PRIM` funktioniert, aber bei dem die `while`-Schleife so abgeändert ist, dass die Variable `:IT` nicht mehr alle Zahlen durchläuft, die kleiner sind als `:NUM`, sondern nur solche, die *kleiner gleich* $\sqrt{\texttt{:NUM}}$ sind.

Hierdurch wird das Programm viel schneller, da es weniger Vergleiche machen muss.

Teste `PRIM 22277` und `PRIM2 22277`.

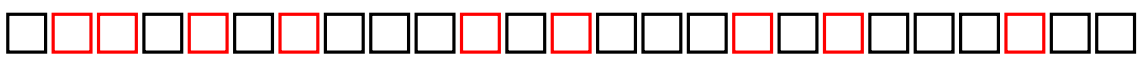
Hinweis: Wir erinnern uns, dass „kleiner gleich“ als „`<=`“ geschrieben wird.

Die Zahl 10007 ist eine Primzahl. Mit der ursprünglichen Version von `PRIM2` würde die `while`-Schleife ungefähr 10000-mal durchlaufen. Mit der neuen Version wird wird dies nur noch ungefähr 100-mal getan, was eine enorme Beschleunigung darstellt.

Das folgende Programm `PRIMZAHLEN` benutzt `PRIM2` als Unterprogramm und stellt die Vorkommen von Primzahlen unter den ersten `:ANZ` natürlichen Zahlen grafisch dar.

```
to PRIMZAHLEN :ANZ
pu lt 90 fd 300 rt 90 pd
make "TEST 1
repeat :ANZ [
  pu rt 90 fd 10 lt 90 pd
  PRIM2 :TEST
  make "TEST :TEST+1
]
end
```

Wenn `PRIMZAHLEN 25` eingegeben wird, so erhalten wir das folgende Bild.

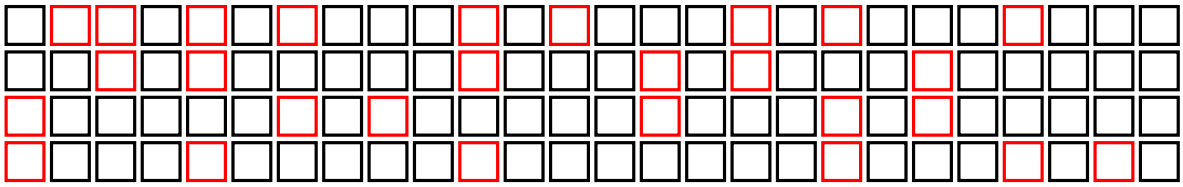


Jetzt können wir die Darstellung noch ein bisschen verfeinern, indem wir die Quadrate in mehrere Zeilen zeichnen. Wir schreiben zu diesem Zweck ein Programm `PRIMZAHLEN2` mit

einem zweiten Parameter :REIHE. Wir lassen die Schildkröte immer an den Anfang der nächsten Zeile gehen, wenn eine durch :REIHE teilbare Anzahl von Quadraten gezeichnet wurde.

```
to PRIMZAHLEN2 :ANZ :REIHE
pu lt 90 fd :REIHE*10/2 rt 90 pd
make "TEST 1
repeat :ANZ [
  pu rt 90 fd 10 lt 90 pd
  PRIM2 :TEST
  make "REST mod :TEST :REIHE
  if :REST = 0 [
    pu lt 90 fd :REIHE*10 lt 90 fd 10 rt 180 pd
  ] [ ]
  make "TEST :TEST+1
]
end
```

Mit `PRIMZAHLEN2 104 26` erhalten wir beispielsweise das folgende Bild.



Nun widmen wir uns der Hauptaufgabe dieses Kapitels.

Schritt 4 (Primzahlpotenzen)

Eine Primzahlpotenz ist eine natürliche Zahl, die genau einen Primfaktor besitzt. Die Zahl 27 ist also beispielsweise eine solche Zahl, denn sie hat die Primzahlzerlegung

$$27 = 3 \cdot 3 \cdot 3 = 3^3.$$

Offensichtlich ist damit auch jede Primzahl eine Primzahlpotenz.

In diesem Projekt sollst du ein Programm PRIMPOT erstellen, das testet, ob eine Zahl eine Primzahlpotenz ist.

Hierbei sollst du folgendermassen vorgehen.

1. Schreibe das Programm PRIM so zu einem Programm PRIMOUT um, dass es den Befehl **output** benutzt, anstatt Quadrate zu zeichnen. Ist der Wert von :NUM eine Primzahl, soll eine 1 zurückgegeben werden, sonst eine 0.
2. PRIMPOT hat einen Parameter :TEST. Für den übergebenen Wert dieses Parameters wollen wir herausfinden, ob er eine Primzahlpotenz ist.
3. Es testet zuerst mit PRIMOUT, ob der Wert von :TEST eine Primzahl ist. Ist dieser Test erfolgreich, wird „Primzahl.“ ausgegeben und das Programm mit **stopall** beendet.

   ```
   make "ISTPRIM PRIMOUT :TEST
   if :ISTPRIM=1 [pr [Primzahl.] stop] []
   ```

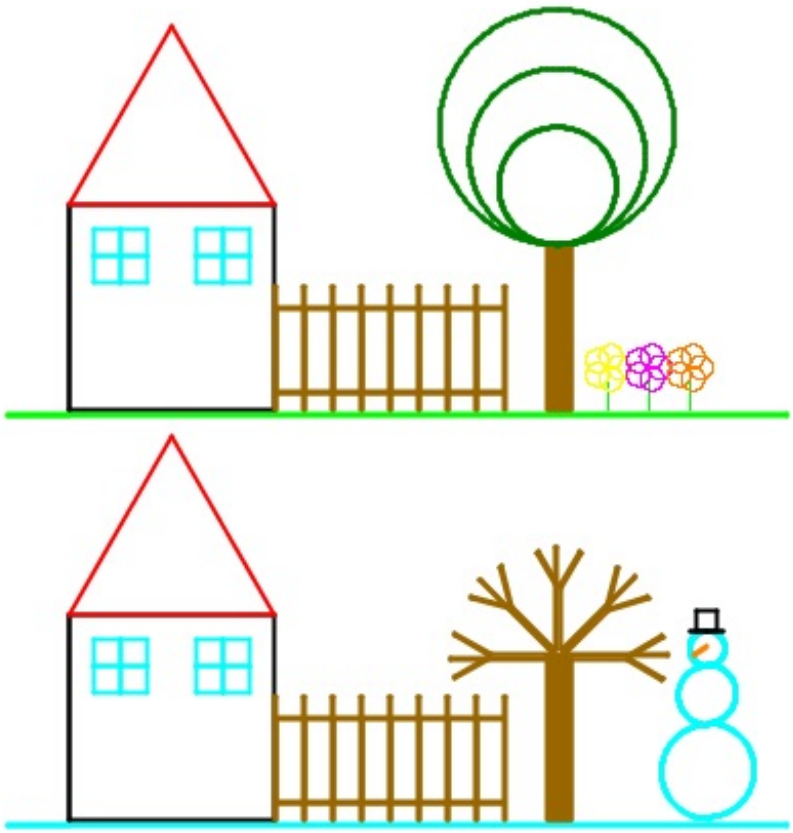
4. Sonst wird jede Zahl, die kleiner als der Wert von :TEST ist, aufgezählt und mit PRIMOUT überprüft, ob diese Zahl eine Primzahl ist. Ist dies der Fall, wird überprüft, ob sie den Wert von :TEST teilt.
5. Wird eine erste solche Primzahl gefunden, merkt PRIMPOT sich dies, indem es den Wert einer Variablen :FUND auf 1 setzt. Am Anfang des Programms wird :FUND auf 0 gesetzt. Wird eine zweite solche Zahl (ein zweiter Primfaktor) gefunden, so wird dies festgestellt, weil der Wert von :FUND bereits gleich 1 ist. In diesem Fall wird „Mehr als ein Teiler.“ ausgegeben und das Programm wieder mit **stopall** beendet.
6. Ist am Ende :FUND noch immer gleich 1, so wird „Primzahlpotenz. Basis:“ und die Primzahl, die den Wert von :TEST teilt, ausgegeben.

Teste PRIMPOT mit kleinen Werten.

Meine Notizen

Befehlsübersicht

`fd 100`	100 Schritte vorwärts gehen
`bk 50`	50 Schritte rückwärts gehen
`cs`	alles löschen und neu beginnen
`rt 90`	90 Grad nach rechts drehen
`lt 90`	90 Grad nach links drehen
`repeat 4 [...]`	das Programm in `[...]` wird viermal wiederholt
`pu`	die Schildkröte wechselt in den Wandermodus
`pd`	die Schildkröte wechselt zurück in den Stiftmodus
`setpc 3`	wechselt die Stiftfarbe auf die Farbe 3
`setpw 5`	wechselt den Stift-Durchmesser auf 5
`to NAME`	erstellt ein Programm mit einem Namen
`to NAME :PARAMETER`	erstellt ein Programm mit einem Namen und einem Parameter
`end`	alle Programme mit einem Namen enden mit diesem Befehl
`pe`	die Schildkröte wechselt in den Radiergummimodus
`ppt`	die Schildkröte wechselt zurück in den Stiftmodus
`wait 5`	die Schildkröte wartet 5 Zeiteinheiten
`make "VARIABLE 10`	setzt den Wert einer Variablen auf 10
`sqrt :VARIABLE`	berechnet die Quadratwurzel einer Variablen
`pr :VARIABLE`	gibt den aktuellen Wert einer Variablen aus
`if :X=1 [P1] [P2]`	führt `P1` aus, wenn `:X` gleich 1 ist und sonst `P2`
`stop`	beendet das aktuelle Programm oder die aktuelle Schleife
`stopall`	beendet alle Programme
`mod 13 5`	gibt den Rest bei Division von 13 durch 5, also 3, aus
`while [:VARIABLE>1] [...]`	das Programm in `[...]` wird wiederholt solange der Wert von `:VARIABLE` grösser 1 ist
`output :VARIABLE`	gibt den Wert von `:VARIABLE` zurück
`home`	setzt die Schildkröte wieder auf die Startposition



Programmieren mit LOGO

Projekte

Informationstechnologie und Ausbildung
ETH Zürich, CAB F 15.1
Universitätstrasse 6
CH-8092 Zürich

www.ite.ethz.ch
www.abz.inf.ethz.ch