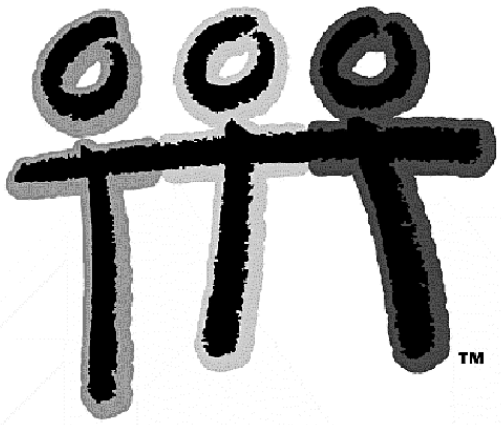


T³ EUROPE

Funktionen und Programme auf dem *TI-92* und *TI-89*

Dr. Thomas Himmelbauer

Ein Unterrichtsbehelf zum Einsatz moderner Technologien im Mathematikunterricht

T³ Österreich / ACDCA am PI-Niederösterreich, Hollabrunn

Inhalt

Vorwort

Die Grundzüge dieses Skriptums entstammen Unterlagen, die ich für ein 3-tägiges T^3-Seminar mit dem Thema „Programmierung am TI-92“ im Herbst 1999 in Wien verfasst habe. Diese Unterlagen wurden durch meine Erfahrungen in der Lehrerausbildung und aus dem Unterricht mit meinen Schülern ergänzt. Beruflich bin ich als Lehrer für Mathematik und Physik am Gymnasium Neulandschule in Wien tätig.

Die Inhalte dieses Skriptums sind auf dem TI-92, dem TI-92 PLUS und dem TI-89 (lediglich die andere Bildschirmauflösung muss berücksichtigt werden) anwendbar.

Das Beherrschen der Programmierung des TI-92/89 durch den Lehrer erscheint mir aus mehreren Gründen dringend notwendig.

- Der Lehrer sollte einen Überblick über die Möglichkeiten der Programmierung haben, um abschätzen zu können, was Schüler durch selbstgeschriebene Programme abkürzen oder vereinfachen können.
- Der Lehrer sollte am Programmieren interessierten Schülern bei ihren Problemen zumindestens etwas helfen können.
- Das Erstellen einfacher Programme und Funktionen sollte Teil des Unterrichts sein, damit die Möglichkeiten der Geräte wirklich ausgenützt werden.
- Der Lehrer sollte einfache Übungsprogamme erstellen können, mit denen die Schüler schon gelernte Inhalte üben können.
- Der Lehrer sollte Demonstrationsprogramme erstellen können, mit denen er den Schülern neue Inhalte besser erklären kann.

Unterstützung für die ersten beiden Punkte findet man in allen Kapiteln des vorliegenden Lehrbehelfs. Dabei sollten Anfänger mit dem ersten Kapitel beginnen, da das Skriptum aufbauend verfasst ist und der Schwierigkeitsgrad der Programmierung stetig wächst. Auf den dritten Punkt gehen vor allem die Kapitel 1.3, 1.4 und 1.5 und 2.2, 2.3 und 2.4 ein. Der vierte Punkt wird vor allem in den Kapiteln 2.6 und 2.7 behandelt, während auf den letzten Punkt in den Kapiteln 2.5 und 2.8 besonders eingegangen wird.

Das Skriptum versucht alle möglichen Arten von Steuerung der Eingabe und Ausgabe zu beschreiben und ist auch nach diesen Steuerungsmöglichkeiten strukturiert.

Weitere Anregungen und fertige Programme von mir zum Thema Demonstration und Übung findet man auf auf der Homepage der ACDCA (Austrian Center for the Didactics of Computer Algebra) www.acdca.ac.at bei den Unterrichtsmaterialien im Unterabschnitt Programme.

Interaktive Übungsprogramme zum Rechnen mit Termen, zur Darstellung von linearen Funktionen und zum Differenzieren und Integrieren findet man in der bk-teachware Schriftenreihe Nr. SR-15: Mathe Trainer I von Josef Böhm.

Mein besonderer Dank gilt Josef Böhm für seine Anregungen zum Inhalt und das Erstellen des endgültigen Layouts.

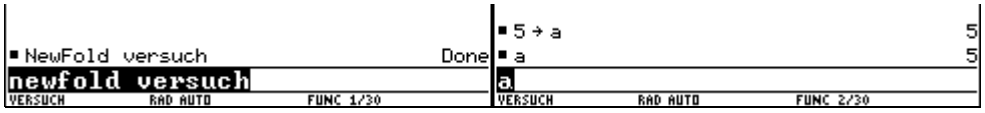
Thomas Himmelbauer

1 Funktionen

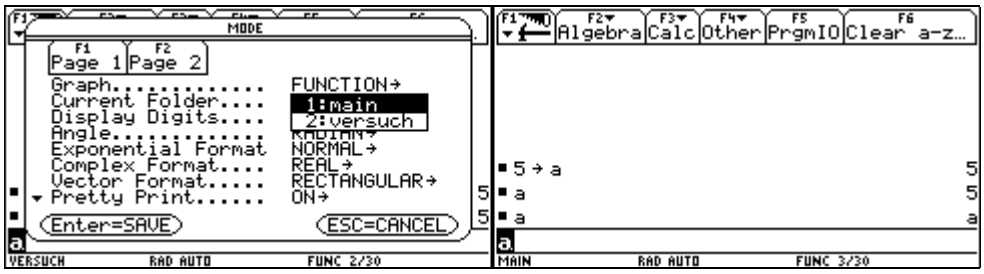
1.1 Systemvariable, Foldervariable, lokale Variable, globale Variable:

Systemvariable und Foldervariable:

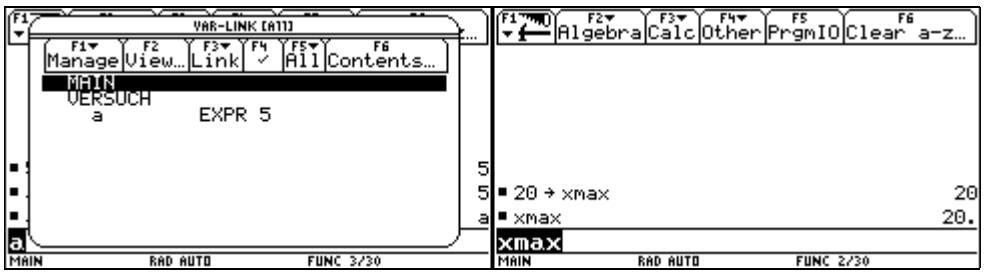
Zunächst legen wir mit dem Befehl **newfold** einen neuen Folder versuch an und speichern 5 unter der Variablen a ab. Zur Kontrolle rufen wir die Variable *a* noch einmal auf.



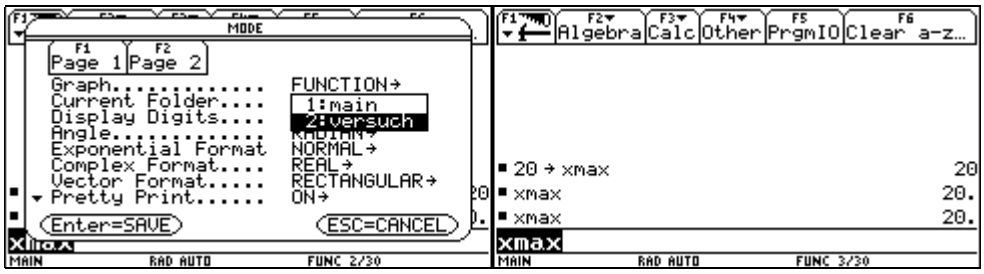
Dann wechseln wir über [MODE] in den Folder main. Der Aufruf von a zeigt, dass die Variable *a* unbelegt ist. Jeder vom Benutzer verwendete Variablenname ist nur für den jeweiligen Folder gültig. In jedem anderen Folder ist dieser Variablenname noch unbelegt und kann auch anders belegt werden.



Dies kann auch über [VAR-LINK]überprüft werden. Nur im Folder versuch gibt es momentan eine Variable a. Speichern wir nun den Wert 20 unter der **Systemvariablen** xmax (Windowvariable) im Folder main ab und kontrollieren wir den Vorgang durch erneutes Aufrufen von xmax.



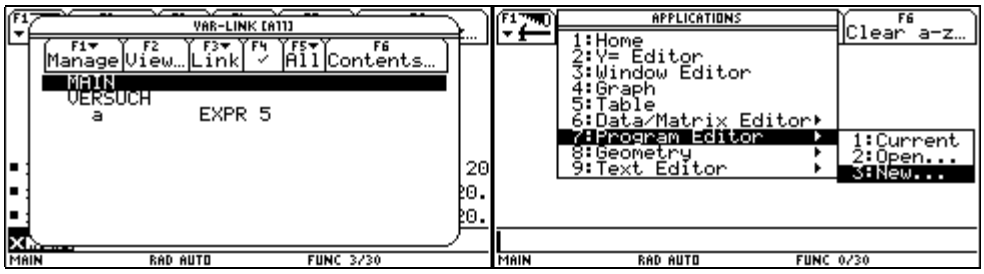
Dann wechseln wir über [MODE] in den Folder versuch und rufen xmax auf. Auch in diesem Folder hat xmax den Wert 20. Der Wert einer Systemvariablen ist unabhängig vom aktuellen Folder. Eine Aufzählung aller Systemvariablen befindet sich im Handbuch auf Seite 491.



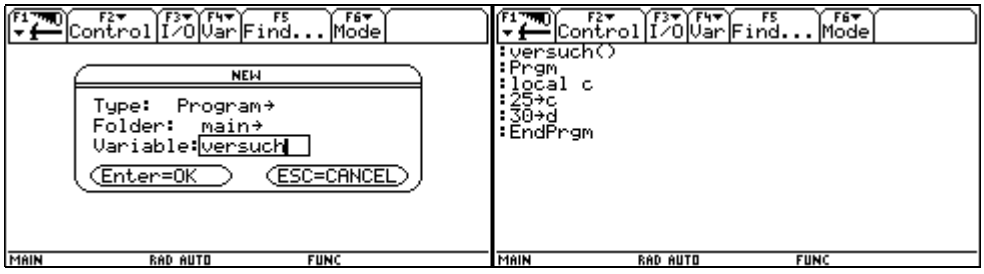
Im [VAR-LINK]-Fenster erscheinen die Systemvariablen nicht.

Globale und lokale Variable:

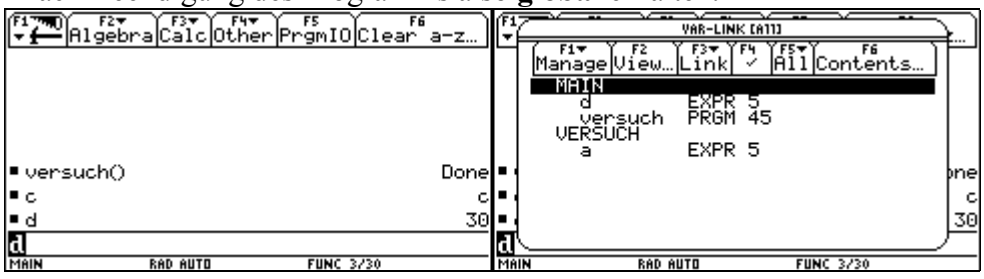
Diese Unterscheidung ist für Programme und Funktionen wichtig. Wir öffnen den Programmeditor über [APPS] 7 und 3 und legen dort ein neues Programm mit dem Namen versuch an.



Das Programm soll nur zum Speichern von Zahlenwerten dienen. Wir definieren die Variable *c* als **Local** und speichern 25 unter *c* und 30 unter *d* ab.



Wir kehren mit [2nd][ESC] in den Homebereich zurück. Dann wird das Programm versuch() gestartet (- nicht auf die Klammern vergessen). Danach kontrollieren wir die Belegung von *c* und *d*. Die Variable *c* ist unbelegt, sie wurde nur innerhalb des Programms, also lokal verwendet. Die Variable *d* ist belegt. Ihre Belegung bleibt auch nach Beendigung des Programms also **global** erhalten.

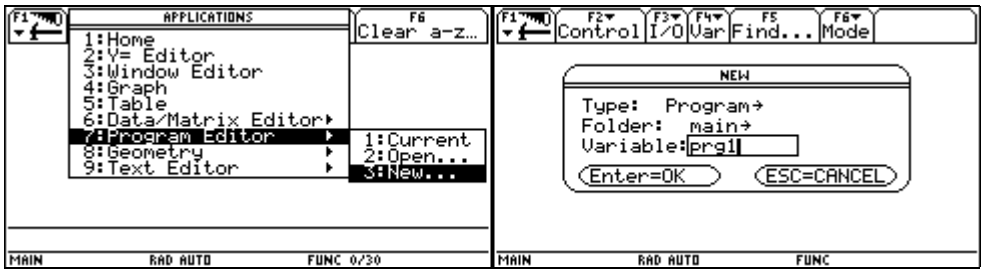


Lehrinhalte: Unterschiede zwischen Systemvariablen und Variablen, die in bestimmten Foldern abgespeichert sind, Unterschiede zwischen lokalen und globalen Variablen bei Verwendung in Funktionen oder Programmen.

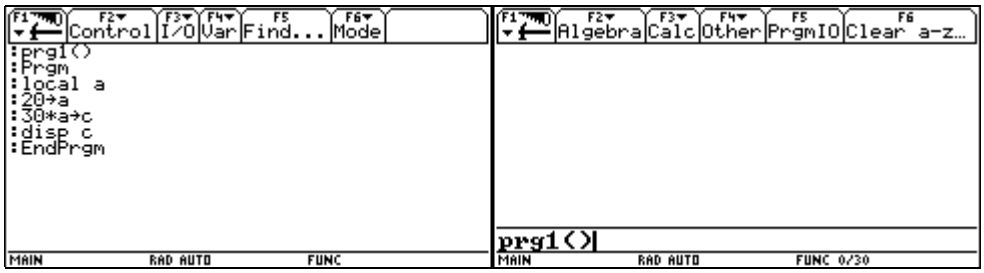
Befehle: NewFold, Local

1.2 Unterschiede zwischen Funktionen und Programmen

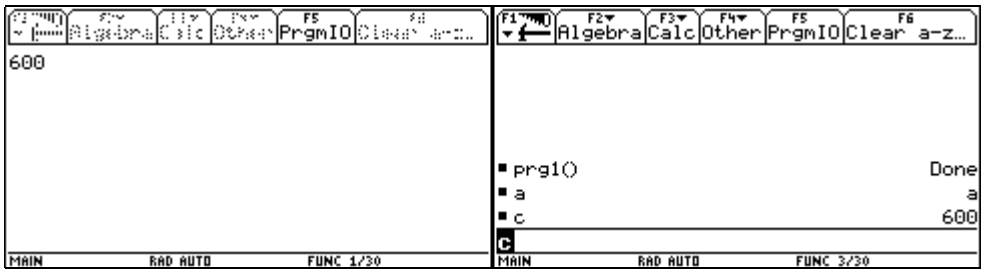
Wir legen im Programm Editor ein neues Program mit dem Namen prg1 an.



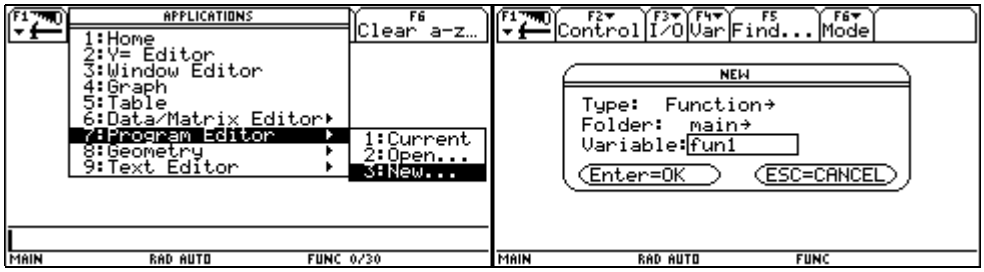
Die Variable *a* wird als **Local** definiert und mit 20 belegt. Danach wird *a* mit 30 multipliziert und unter *c* **global** abgespeichert. Durch den Befehl **Disp** (Display) wird der Wert der Variablen *c* im **Programm Input-Output** Schirm angezeigt. Das ist ein vom Homebereich getrennter Ein- und Ausgabebereich für Programme. Vom Homebereich gelangt man über den Menüpunkt F5 PrgmIO in den Programm Input-Output Schirm. Mit 2nd ESC kehren wir in den Homebereich zurück und starten das Programm prg1().



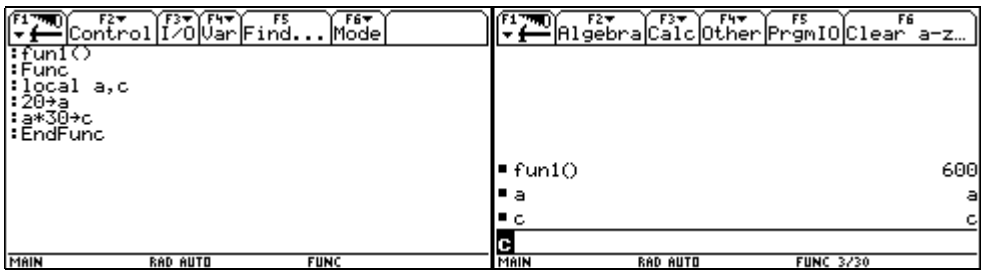
Es meldet sich der Programm Input-Output Schirm mit der Belegung von *c*. Wir wechseln über F5 zurück in den Homebereich und kontrollieren die Belegungen. Die lokale Variable *a* ist unbelegt. Die globale Variable *c* ist belegt. Ein **Programm** kann also **globale und lokale Variable** verwalten . Variable, die nicht als lokal erklärt wurden, werden global verwaltet. **Alle Befehle** des Rechners können in einem **Programm** verwendet werden. Für die **Ausgabe** müssen **eigene Befehle** verwendet werden, sie erfolgt **nicht** im **Homebereich.**



Wenden wir uns nun den Funktionen zu. Wir öffnen wieder in den Programm Editor und bezeichnen einen neue Funktion (umstellen von Type von Program auf Function) mit dem Namen fun1.



In einer **Funktion** sind **nur lokale Variable** erlaubt. Alle vorkommenden Variablen **müssen** als **Local** definiert werden. Wir lassen die gleiche Berechung wie in unserem Programm durchführen. Aber wir benötigen keine eigenen Ausgabefunktionen. Die **Ausgabe** erfolgt immer über **den Homebereich**. Das Ergebnis der letzten Anweisung ist automatisch die Antwort auf einen Funktionsaufruf. Nach dem Wechsel in den Homebereich starten wir unsere Funktion (Klammern nicht vergessen) und erhalten als Antwort 600. Die Variablen *a* und *c* sind unbelegt. In einer **Funktion** dürfen **nicht alle Befehle** des Rechners verwendet werde. Eine genaue Auflistung findet man im Handbuch auf Seite 303. Stark vereinfacht kann man sagen: eine Funktion darf verzweigte Anweisungen und Schleife, aber praktisch keine graphischen Befehle enthalten.



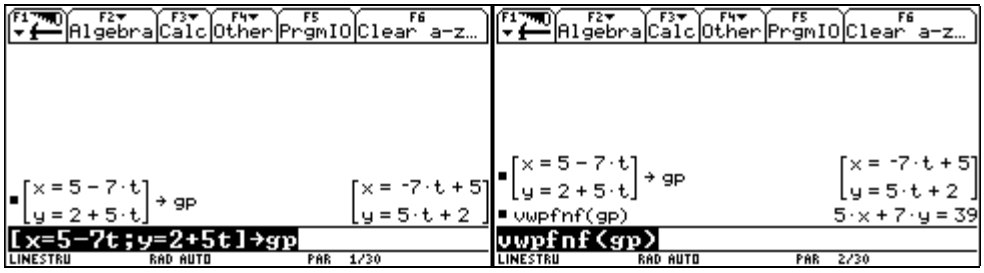
Lehrinhalte: Unterschiede zwischen Funktionen und Programmen.

1.3 Funktionen mit linearer Struktur

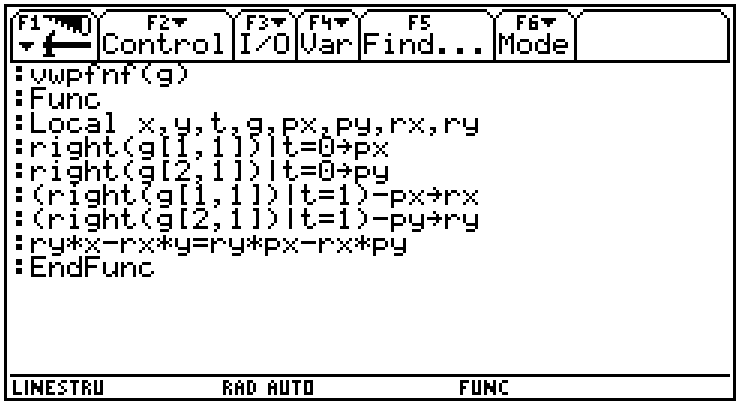
Funktion: Verwandlung einer Geradengleichung von Parameterform zur Hauptform

Benutzung:

Zunächst wollen wir uns die Verwendung der fertigen Funktion ansehen. Nachdem man sich versichert hat, dass die Variablen *x*, *y* und *t* nicht belegt sind, gibt man die Parameterform einer Geradengleichung als Matrix mit einer Spalte und zwei Zeilen ein. Damit kommt man der üblichen Darstellung am nächsten. Man speichert die Gleichung unter dem Variablennamen gp ab. Dann ruft man die Funktion vwpfnf auf und übergibt die Variable *gp* als Parameter innerhalb der runden Klammern. Als Antwort erhält man die Gleichung in Normalvektorform.

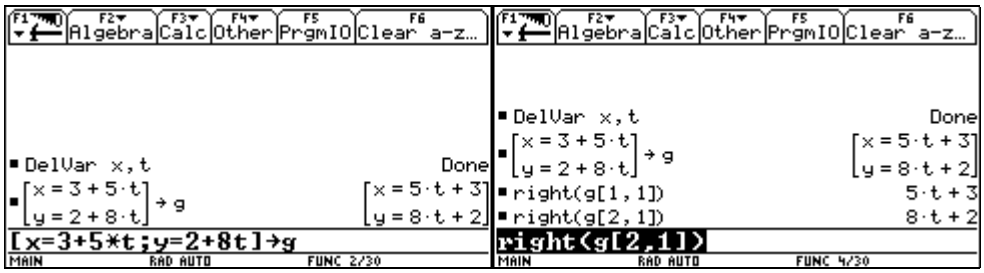


Programmcode:

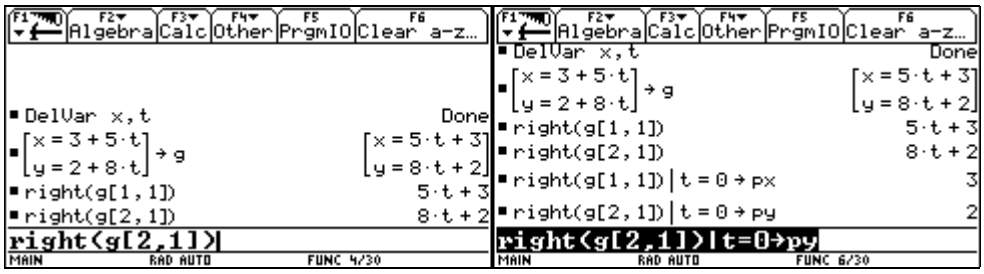


Erläuterung der auftretenden Befehle:

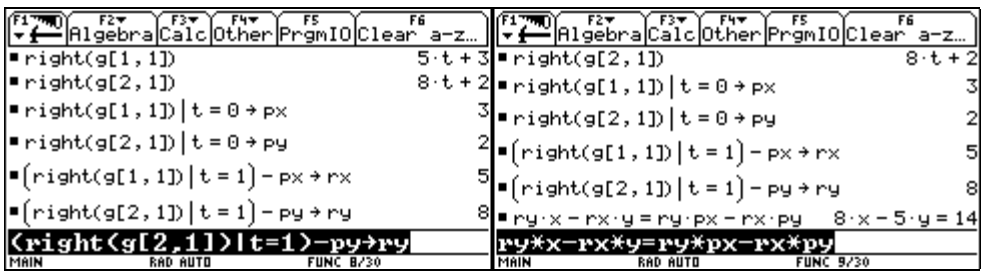
Sollte *x,y* oder *t* belegt sein, dann löschen wir mit **DelVar** Variablenname die Belegung. Dann geben wir eine Parametergleichung unter g ein. Eine **Matrix** wird mit einer eckigen Klammer begonnen und beendet, die Zeilen werden durch einen Strichpunkt getrennt. Mit Hilfe von g[1,1] bzw. g[2,1] wird auf die *x*- bzw. *y*-Koordinate der Gleichung *g* zugegriffen. Mit der Funktion right erhält man die rechte Seite einer Gleichung.



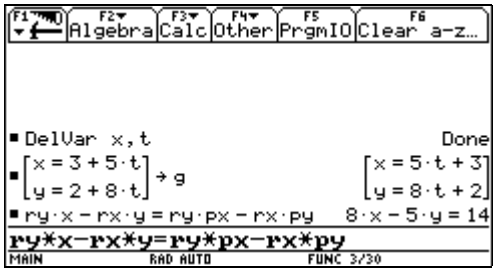
Wertet man die rechten Gleichungsseiten mit dem **With-Operator** ([|] = 2nd (K)=) für $t = 0$ aus, so erhält man die Koordinaten des Punktes $P(px, py)$ auf der Geraden *g*.



Wertet man die rechten Gleichungsseiten mit dem **With-Operator** für $t = 1$ aus und subtrahiert man die Punktekoordinaten, so erhält man die Koordinaten des Richtungsvektors $\begin{pmatrix} rx \\ ry \end{pmatrix}$ der Gleichung. Nun kann die Normalvektorform der Gleichung aufgestellt werden.



Zum Abschluß zur Kontrolle noch ein Vergleich mit der Parameterdarstellung.



Lehrinhalte: lokale Variable, Eingabe ganzer Gleichungen vom Homebereich aus, Ausgabe in den Homebereich, Matrizen

Befehle: **Local, right**, With-Operator

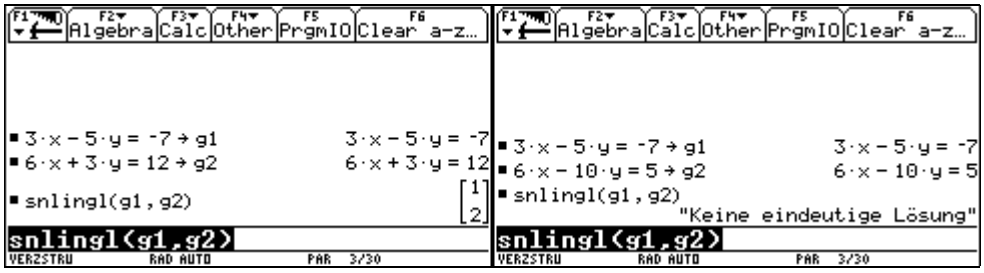
Übung: Verwandlung einer Geradengleichung von der Anstieg-Abschnittsform zur Parameterform

1.4 Funktionen mit verzweigter Struktur

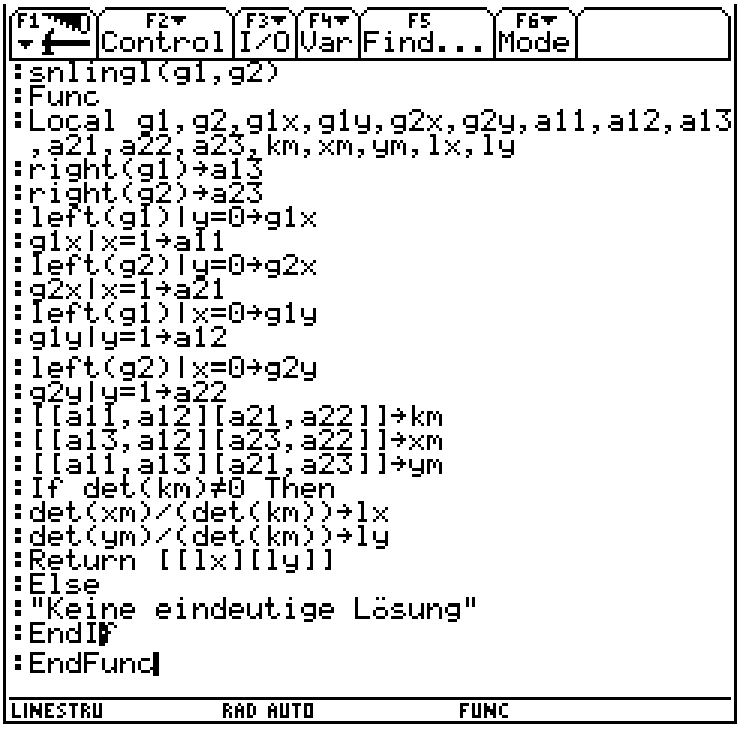
Funktion: Lösung von zwei linearen Gleichungen mit zwei Unbekannten

Benutzung:

Man speichert zwei lineare Gleichungen mit zwei Unbekannten unter g1 und g2 ab. Dann ruft man die Funktion snlingl auf und übergibt die beiden Gleichungen als Parameter. Als Antwort erhält man entweder die eindeutige Lösung als Lösungsvektor oder den Text "Keine eindeutige Lösung", falls keine oder unendlich viele Lösungen vorliegen.

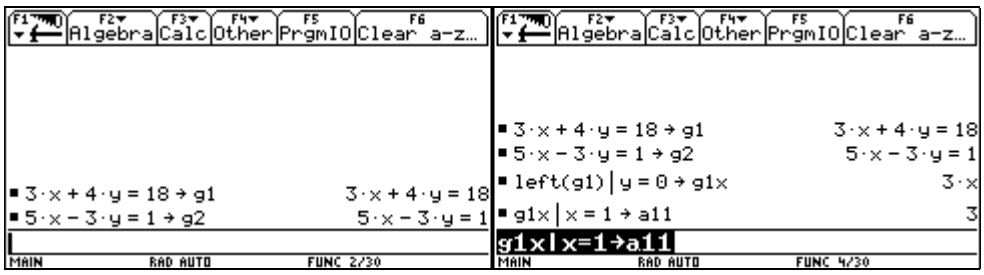


Programmcode:

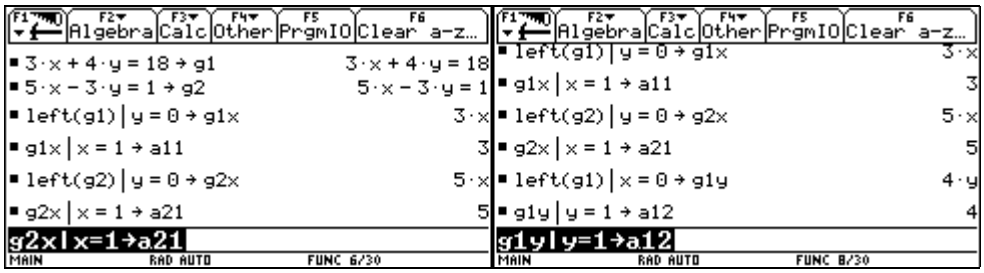


Erläuterung der auftretenden Befehle:

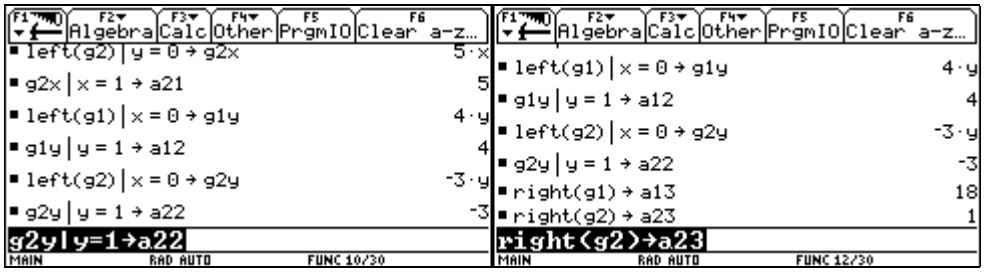
Zunächst werden die beiden Gleichungen als g1 und g2 abgespeichert. Durch den Befehl **left**(g1) wird auf die linke Seite der 1. Gleichung zugegriffen und mit dem **|-Operator** durch die Belegung von *y* mit 0 der *x*-hältige Anteil der linken Gleichungsseite ermittelt. Durch Belegung von *x* mit 1 durch den [|]-Operator, wird der *x*-Koeffizient *a11* der 1. Gleichung ermittelt.



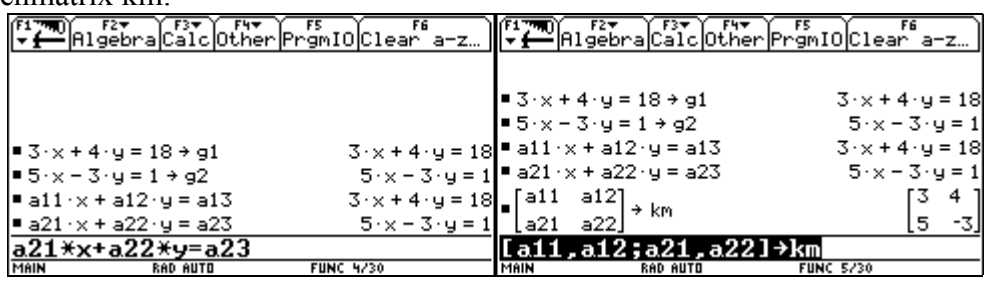
In analoger Weise bestimmt man den *y*-Koeffizienten der 1. Gleichung *a12* und die Koeffizienten von *x* und *y* der 2. Gleichung *a21* bzw. *a22*.



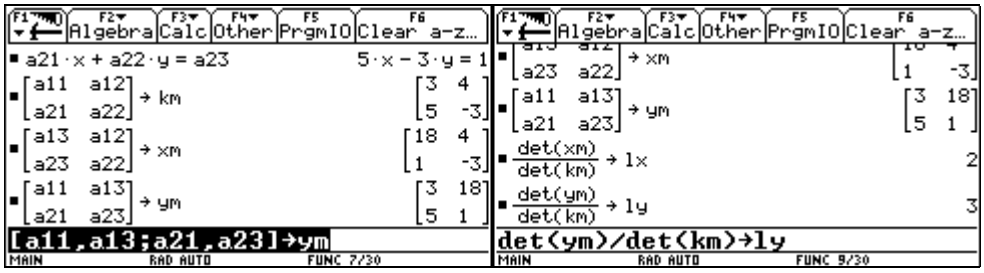
Wiederum mit der Funktion **right** können die Konstanten der Gleichungen unter *a13* und *a23* gespeichert werden.



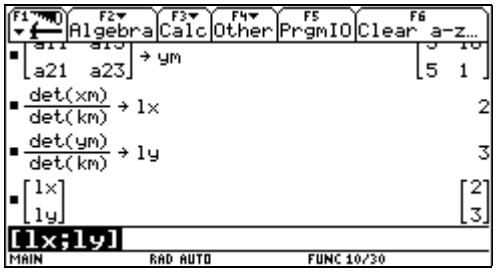
Wir können unsere bisherige Berechnung durch Eingabe der Gleichungen mit Hilfe der selektierten Koeffizienten überprüfen. Die Lösung wollen wir über die Cramersche Regel erhalten. Daher stellen wir die entsprechenden **Matrizen** auf. Zuerst die Koeffizientenmatrix km:



Dann die *x*-Matrix xm und die *y*-Matrix ym. Danach werden mit Hilfe der **Determinantenfunktion** det die Lösungen lx und ly berechnet.



Da diese Berechnung nur möglich ist, wenn die Determinante der Koeffizientenmatrix ungleich Null ist, ist mit Hilfe der If-Anweisung in dieser Funktion eine Verzweigung eingebaut. Wenn die Determinante ungleich Null ist, dann wird die Lösung berechnet (Then), ansonsten (Else) erfolgt die Ausgabe des Textes "Keine eindeutige Lösung". Der Text muß unter " gesetzt werden, damit er nicht als Variablenname gedeutet wird, was bei dieser Länge des Textes mit einer Fehlermeldung gedankt wird. Um nun auch in jenem Ast der Verzweigung, der nicht am Schluss der Funktion steht, eine Ausgabe zu erreichen muss der Befehl **Return** vor den Lösungsvektor gesetzt werden.



Lehrinhalte: Determinanten, Matrizen, Verzweigungen, Ausgabe in den Homebereich bei Verzweigungen, Ausgabe von mehreren Ergebnissen zugleich in den Homebereich, Ausgabe von Texten in den Homebereich

Befehle: Local, right, left, Matrizen, det, If-Then-Else-EndIf, Return

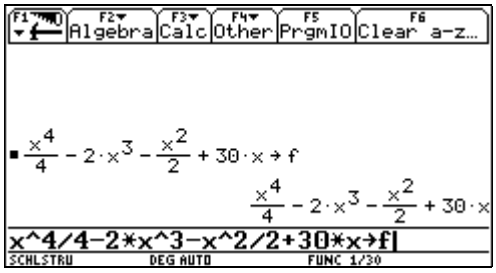
Übung: Schnitt einer Geraden mit einem Kreis

1.5 Funktionen mit Schleifen

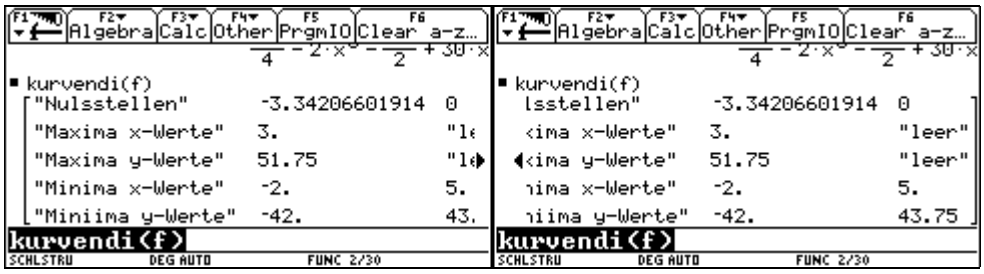
Funktion: Kurvendiskussion einer Polynomfunktion (Nullstellen und Extremwerte)

Benutzung:

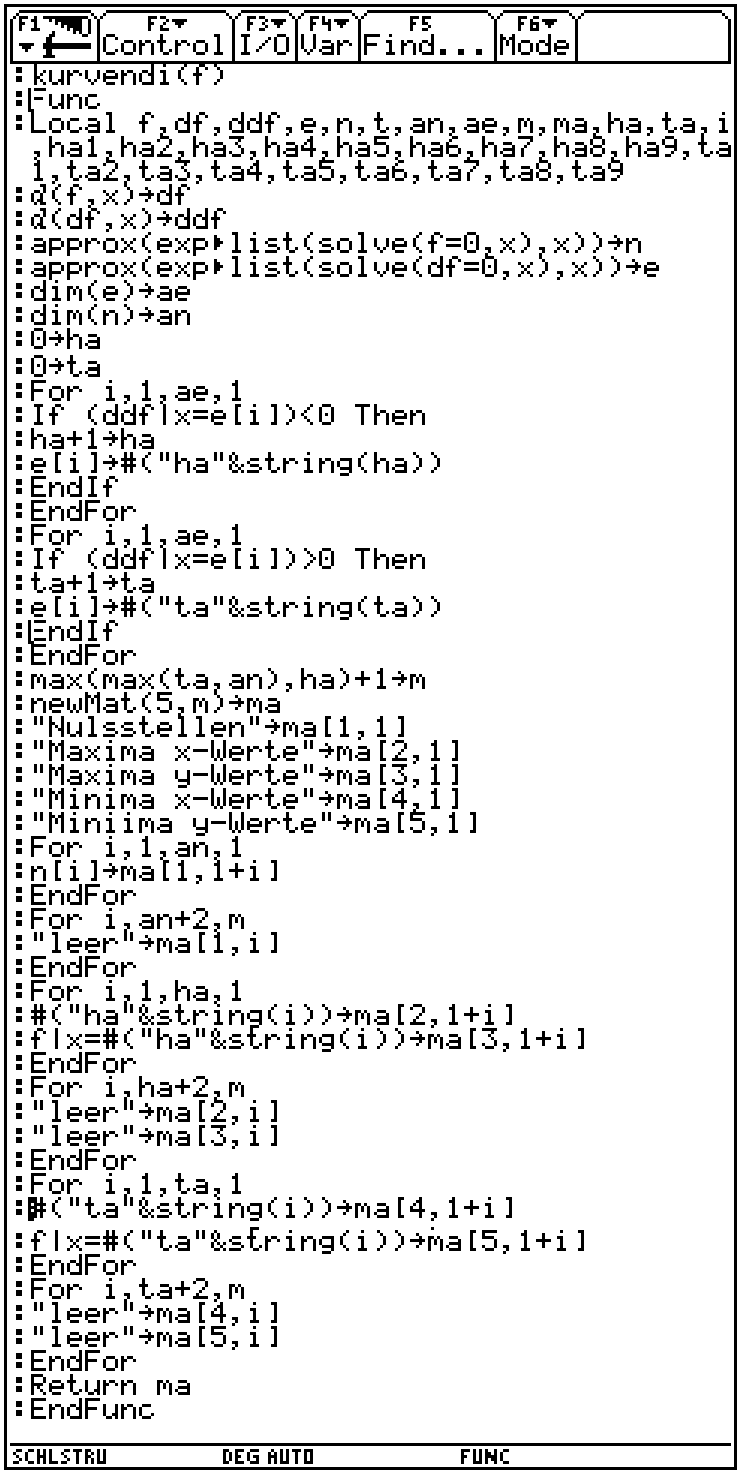
Zunächst wird der zu untersuchende Funktionsterm unter f abgespeichert.



Dann wird f der Funktion kurvendi(f) als Parameter übergeben. Die Antwort ist eine Matrix, in der sich die Werte der Nullstellen und Extremwerte befinden.



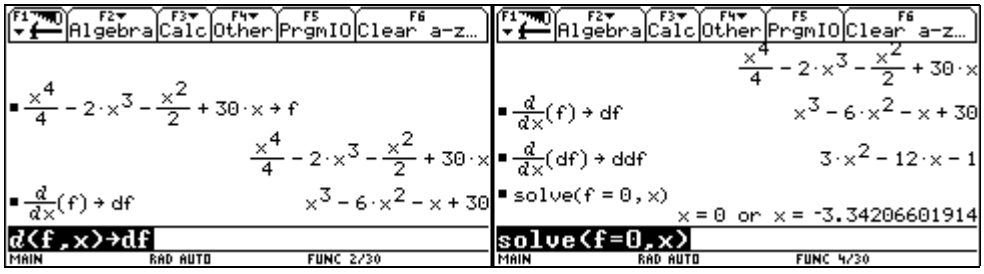
Programmcode:



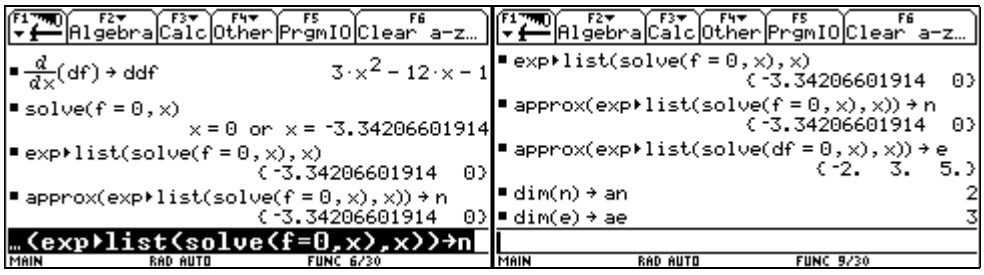
```
F1 F2 Control F3 I/O F4 Var F5 Find... F6 Mode
:kurvendi(f)
:Func
:Local f,df,ddf,e,n,t,an,ae,m,ma,ha,ta,i
 ,ha1,ha2,ha3,ha4,ha5,ha6,ha7,ha8,ha9,ta
 1,ta2,ta3,ta4,ta5,ta6,ta7,ta8,ta9
:d(f,x)→df
:d(df,x)→ddf
:approx(exp▸list(solve(f=0,x),x))→n
:approx(exp▸list(solve(df=0,x),x))→e
:dim(e)→ae
:dim(n)→an
:0→ha
:0→ta
:For i,1,ae,1
:If (ddf|x=e[i])<0 Then
:ha+1→ha
:e[i]→#("ha"&string(ha))
:EndIf
:EndFor
:For i,1,ae,1
:If (ddf|x=e[i])>0 Then
:ta+1→ta
:e[i]→#("ta"&string(ta))
:EndIf
:EndFor
:max(max(ta,an),ha)+1→m
:newMat(5,m)→ma
:"Nulsstellen"→ma[1,1]
:"Maxima x-Werte"→ma[2,1]
:"Maxima y-Werte"→ma[3,1]
:"Minima x-Werte"→ma[4,1]
:"Miniima y-Werte"→ma[5,1]
:For i,1,an,1
:n[i]→ma[1,1+i]
:EndFor
:For i,an+2,m
:"leer"→ma[1,i]
:EndFor
:For i,1,ha,1
:#("ha"&string(i))→ma[2,1+i]
:f|x=#("ha"&string(i))→ma[3,1+i]
:EndFor
:For i,ha+2,m
:"leer"→ma[2,i]
:"leer"→ma[3,i]
:EndFor
:For i,1,ta,1
:#("ta"&string(i))→ma[4,1+i]
:f|x=#("ta"&string(i))→ma[5,1+i]
:EndFor
:For i,ta+2,m
:"leer"→ma[4,i]
:"leer"→ma[5,i]
:EndFor
:Return ma
:EndFunc
SCHLSTRU DEG AUTO FUNC
```

Erläuterung der auftretenden Befehle:

Zunächst speichern wir den Funktionsterm unter f ab. Danach lassen wir mit dem **Befehl [*d*]** die 1. Ableitung df und die 2. Ableitung ddf berechnen. Mit **solve** erhalten wir die Nullstellen.

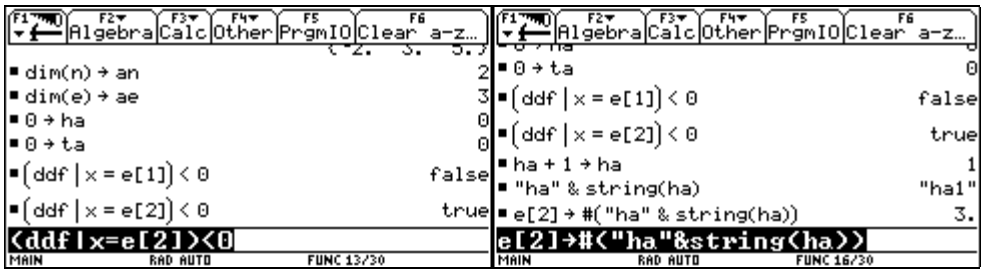


Zur weiteren Berechnung ist es aber notwendig von dieser "hübschen" Schreibweise der Lösungen abzugehen und sie in Listenform umzuformen. Die Funktion **exp▶list** bringt die Lösungen der Größe nach geordnet in eine Liste mit dem Namen n. Mit Hilfe von **approx** erzwingen wir unabhängig von der im MODE festgelegten Einstellung von **Exact/Approx** eine Ausgabe in Dezimalschreibweise. Auch die möglichen Extremwerte werden so ermittelt und kommen in eine Liste mit dem Namen e. Danach wird über die **dim-Funktion** die Anzahl der Nullstellen *an* und die Anzahl der Extremwertkandidaten *ae* bestimmt.

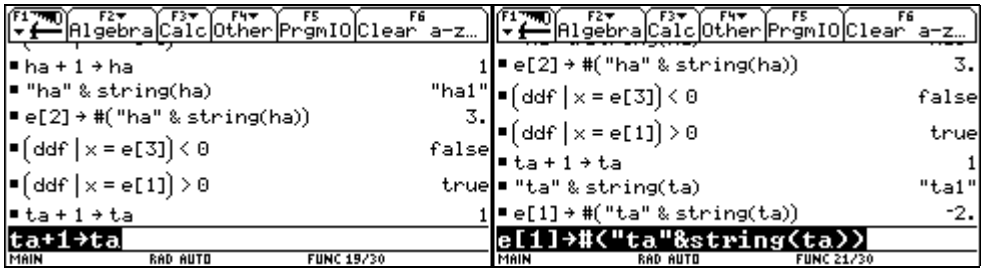


Nun sollen die Minima, Maxima und mögliche Wendepunkte von einander getrennt und die Anzahl der Maxima und Minima *ha* und *ta* bestimmt werden. Zunächst werden diese beiden Anzahlen einmal auf Null initialisiert. Danach werden alle Elemente der Liste der Extremwertkandidaten darauf überprüft, ob sie in die 2. Ableitung eingesetzt ein negatives Vorzeichen besitzen.
Dies geschieht durch den Schleifenbefehl **For i,1,ae,1**. Da die Variable *ae* in unserem Beispiel den Wert 3 besitzt, durchläuft die **Laufvariable** *i* die Werte 1,2,3 usw. bis *ae* **= 3**. Dadurch durchläuft e[i] alle Elemente der Liste e der Extremwerte. Durch eine **If-Abfrage** wird überprüft, ob für ein bestimmtes i_0 das Vorzeichen der zweiten Ableitung negativ ist (in unserem Fall das erste und einzige Mal für e[2], also *i* = 2). In diesem Fall wird die Zählvariable der Maxima *ha* um eins erhöht (in unserem Fall *ha* von Null auf den Wert 1). Das entsprechende Glied $e[i_0]$ der Liste e (in unserem Fall e[2]) wird den Namen ha1, ha2,... erhalten, je nachdem ob es das 1., 2. ....ermittelte Maximum ist. An Hand von *ha1* soll das Generieren von **indizierten Variablen** erläutert werden. Zunnächst wird der **Text = string "ha1"** (die Anführungszeichen dienen dazu einen Text von einer Variablen unterscheiden zu können) gebildet, indem der Text **"ha"** mit dem **Text string(ha)** durch den **&-Operator** (2nd (H)) verknüpft wird. (Wenn z.B. die Variable *ha* den Wert 1 besitzt, erzeugt der Befehl string(ha) die

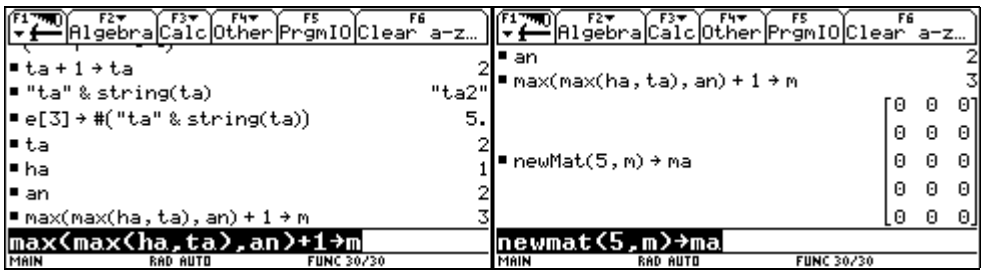
Textkonstante "1".) Danach wird dieser Text "ha1" durch den (Indirection) **#-Operator** (2nd(T)) in die Variable *ha1* umgewandelt. Sie erhält nun den Wert von e[2].



Danach werden alle Elemente der Liste der Extremwertkandidaten überprüft, ob sie eingesetzt in der 2. Ableitung ein positives Vorzeichen besitzen.
Dazu durchläuft die Laufvariable *i* die Werte 1,2,3 usw. bis *ae*. Dadurch durchläuft e[i] alle Elemente der Liste e. Wenn für ein bestimmtes i_0 das Vorzeichen positiv ist (in unserem Fall bei e[1] und e[3], also für $i = 1$ und $i = 3$), wird jedes Mal die Zählvariable der Minima *ta* um eins erhöht (in unserem Fall zunächst auf 1 und dann auf 2). Die entsprechenden Glieder $e[i_0]$ der Liste *e* (in unserem Fall e[1] und e[3], sollen den Namen ta1, ta2... erhalten, je nachdem ob sie dem 1., 2. ....ermittelte Minimum entsprechen. An Hand von *ta1* soll das Generieren indizierter Variabler noch einmal erläutert werden. Zunnächst wird der die **Textkonstante = string "ta1"** gebildet, indem der **Text "ta"** mit dem **Text string(ta)** durch den **&-Operator** verknüpft wird. Danach wird dieser Text "ta1" durch den **#-Operator** in die Variable *ta1* umgewandelt. Sie erhält nun den Wert von e[1] und in der Folge erhält *ta2* den Wert von e[3].

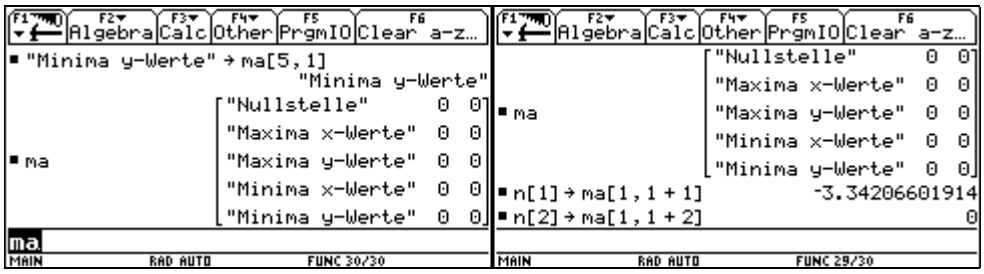


Um die Anzahl der Spalten unserer Ausgabematrix zu bestimmen, bilden wir nun das **Maximum** von *ha*,*ta* mit der Funktion **max(ha,ta)** und erhöhen diese Zahl um 1 für die eine Spalte, die wir für den Text benötigen. Das ergibt insgesamt die Spaltenanzahl *m*. Dann generieren wir mit **newMat(m,5)** eine zunächst nur mit Nullen gefüllte Matrix mit 5 Zeilen *m* Spalten.

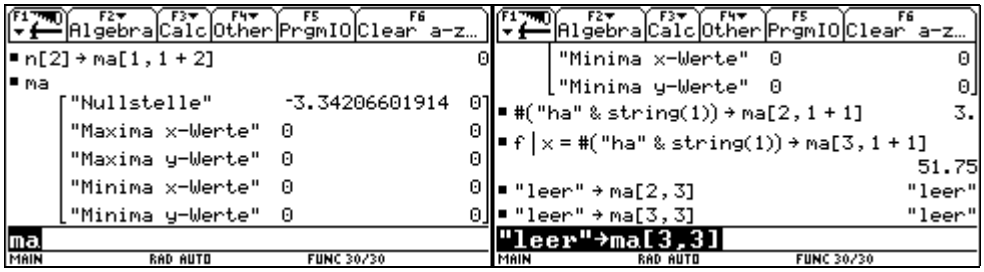


Nun schreiben wir in die 1. Spalte der Matrix die Bezeichnungen der jeweiligen Zeilen.

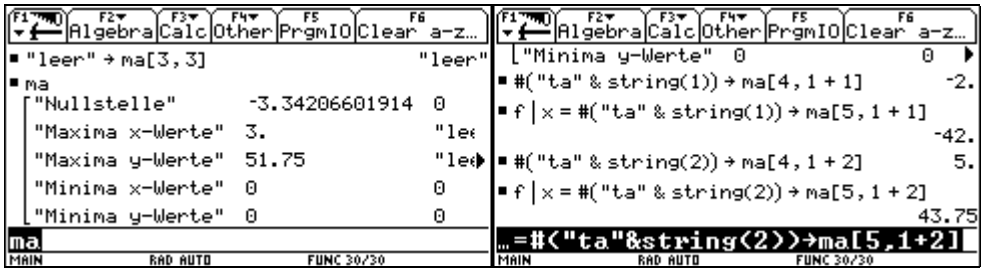
Dann wird über eine Schleife, die 1. Zeile mit den Nullstellen gefüllt.



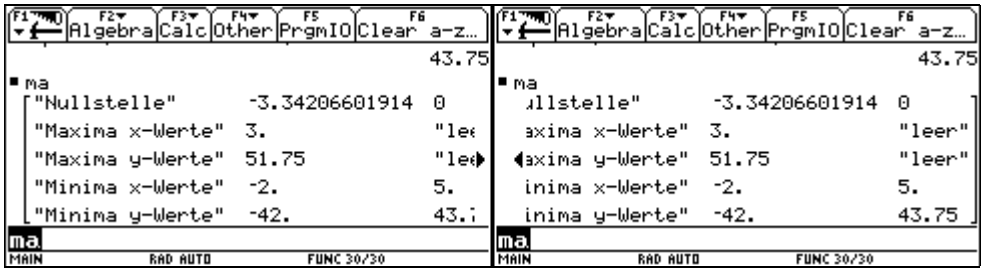
Mit der nächsten Schleife werden in die 2. Zeile die Maxima und in die 3. Zeile deren Funktionswerte gesetzt. Da es nur ein Maximum gibt, müssen die restlichen Plätze dieser Zeilen von den Nullen befreit und mit dem Text "leer" versehen werden.



Mit der letzten Schleife werden in die 4. Zeile die Minima und in die 5. Zeile deren Funktionswerte gesetzt.



Damit ergibt sich die vollständige Ausgabematrix.



Lehrinhalte: Differenzieren, Schleifen, Matrizen, indizierte Variable, Arbeiten mit Strings

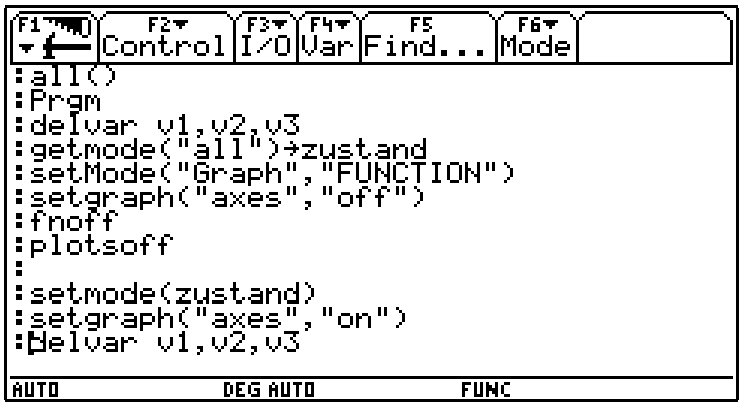
Befehle: For-endFor, solve, dim, max, newMat, exp▶list, string, &,#, *d*(), Return,
If-Then-EndIf,

Übung: Ergänzung der Kurvendiskussion mit Wendepunkten und allfälligen Sattelpunkten

2 Programme

2.1 Allgemeine Grundsätze für TI-Programme

Programm: Musterprogramm, wie Anfang und Ende jedes TI-Programms gestaltet werden könnte.



Zu Beginn sollten alle vorkommenden Variablen mit dem Befehl delvar gelöscht werden. Man weiß nie, welche Variablen der Benutzer schon belegt hat. Danach sollte man die alten [MODE]-Einstellungen des Benutzers mit dem Befehl **getmode** in eine Variable z.B. zustand abspeichern. Dann sind alle Änderungen der [MODE] und Graphik-Einstellungen, die für das Programm nötig sind, mit Hilfe der Befehle setMode und **setGraph** durchzuführen. Abschließend sollte man noch die Plots und Funktionen mit den Befehlen **FnOff** und **PlotsOff** deaktivieren. Falls man den PrgIO verwendet, sollte auch dieser mit **ClrIO** gelöscht werden.

Am Ende sollte man die [MODE]-Einstellungen des Benutzers mit dem Befehl **setMode** wieder herstellen. Gegebenenfalls kann man auch wieder gebräuchliche Graphik-Einstellungen setzen. Alle Variablen, die nicht lokal definiert wurden, oder, die nach Beendigung des Programms nicht mehr benötigt werden, sollen gelöscht werden: Müllvermeidung im [VAR-LINK]-Schirm.

Lehrinhalte: Abspeichern von Rechnereinstellungen, Löschen von Variablen , Funktionen und Plots, Einstellen der Graphikformate

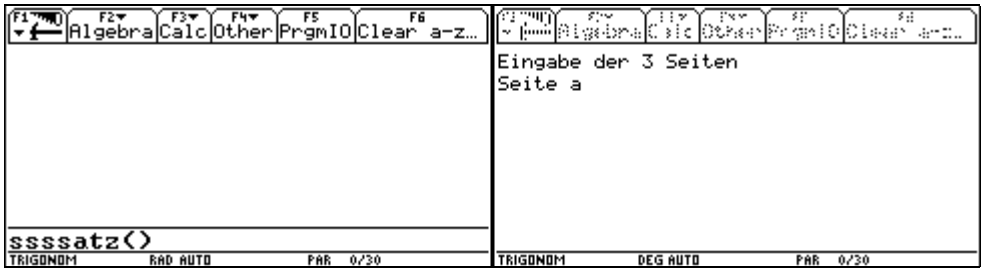
Befehle: getMode, setMode, setGraph, FnOff, PlotsOff

2.2 Steuerung über den Input-Output-Schirm

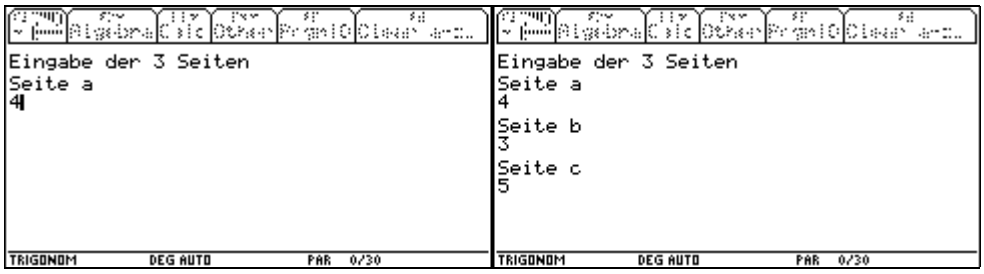
Programm: Berechnung der Winkel eines schiefwinkeligen Dreiecks aus den Seitenlängen

Benutzung:

Nach dem Starten des Programms erscheint der PrgIO mit der Aufforderung die Länge der Seite *a* einzugeben. Nach der Eingabe ist mit ENTER zu bestätigen.



Danach sind auch die Seiten *b* und *c* einzugeben. Nach der Bestätigung von *c* mit ENTER, wird der PrgIO gelöscht und die Ergebnisse der Winkelberechnung angegeben.



Das Pausezeichen rechts unten am Bildschirm deutet einen Programmstop an, der mit ENTER beendet wird. Der Bildschirm wird gelöscht und über F5 kehrt man in den Homebereich zurück.

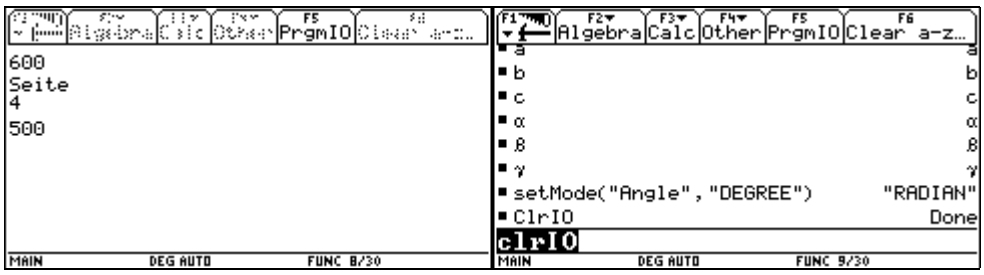
Programmcode:

```
:ssssatz()
:Prgm
:DelVar a,b,c,α,ß,γ
:setMode("Angle","DEGREE")
:ClrIO
:Disp "Eingabe der 3 Seiten"
:Input "Seite a",a
:Input "Seite b",b
:Input "Seite c",c
:ClrIO
:approx(cos⁻¹((a^2-c^2-b^2)/(-2*c*b)))→α
:approx(cos⁻¹((b^2-c^2-a^2)/(-2*c*a)))→ß
:approx(cos⁻¹((c^2-a^2-b^2)/(-2*a*b)))→γ
:Output 10,0,"Winkel α="
:Output 10,60,α
:Output 20,0,"Winkel ß="
:Output 20,60,ß
:Output 30,0,"Winkel γ="
:Output 30,60,γ
:Pause
:ClrIO
:setMode("Angle","RADIAN")
:DelVar a,b,c,α,ß,γ
:EndPrgm
```

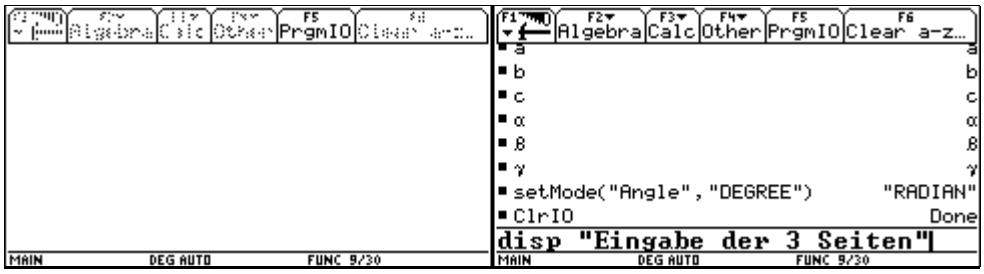
Erläuterung der Befehle:

Zunächst löschen wir die vorkommenden Variablen. Da der MODE für die Winkelmessung (= Angle) auf RADIAN stehen könnnte, wird der MODE dafür mit **setMode** auf DEGREE umgestellt.

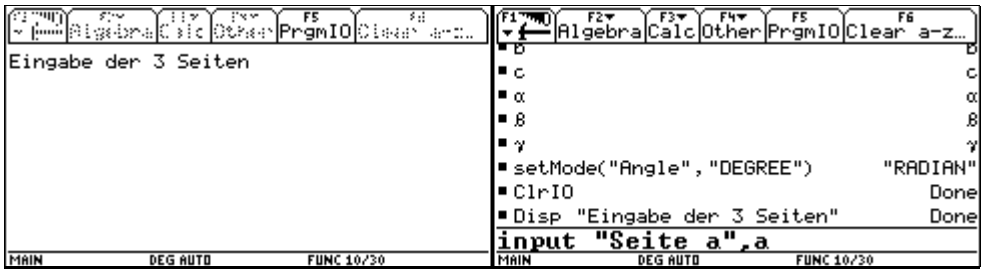
Auch der PrgIO könnte noch beschrieben sein, daher wird er mit ClrIO gelöscht.

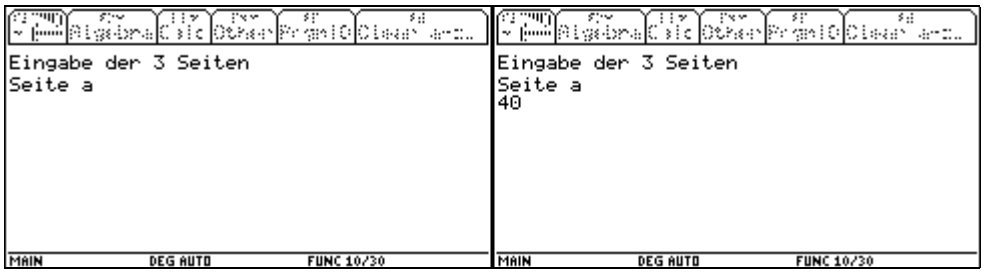


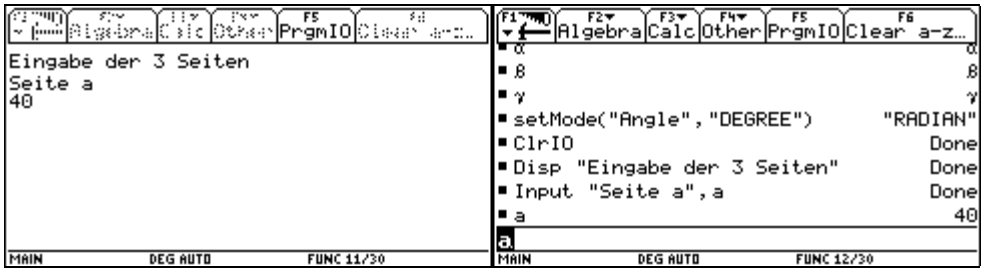
Nun wird mit **Disp** (von display) der Text, "Eingabe der 3 Seiten" in den Programmausgabeschirm geschrieben.

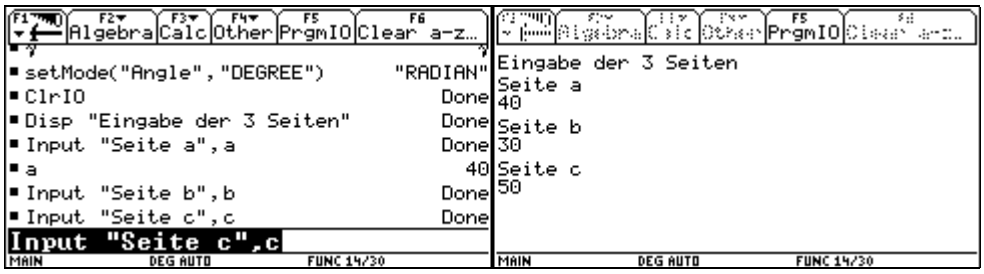


Mit **Input** kann der Text "Seite a" in den PrgIO geschrieben werden. Ein danach vom Benutzer eingegebener Zahlenwert, wird unter der Variablen *a* gespeichert, die dem Text durch einen Beistrich getrennt folgt. Gleiches gilt für die Seiten *b* und *c*.

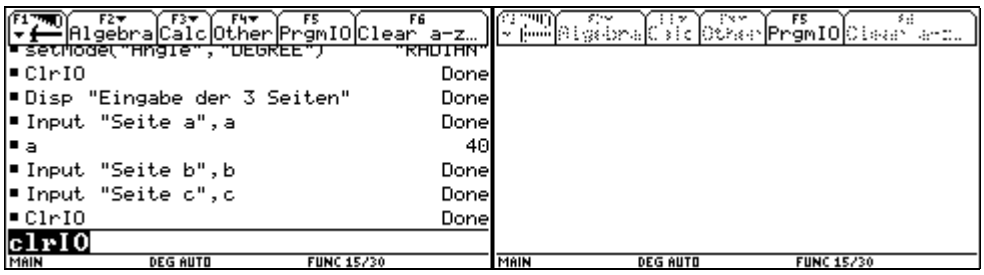




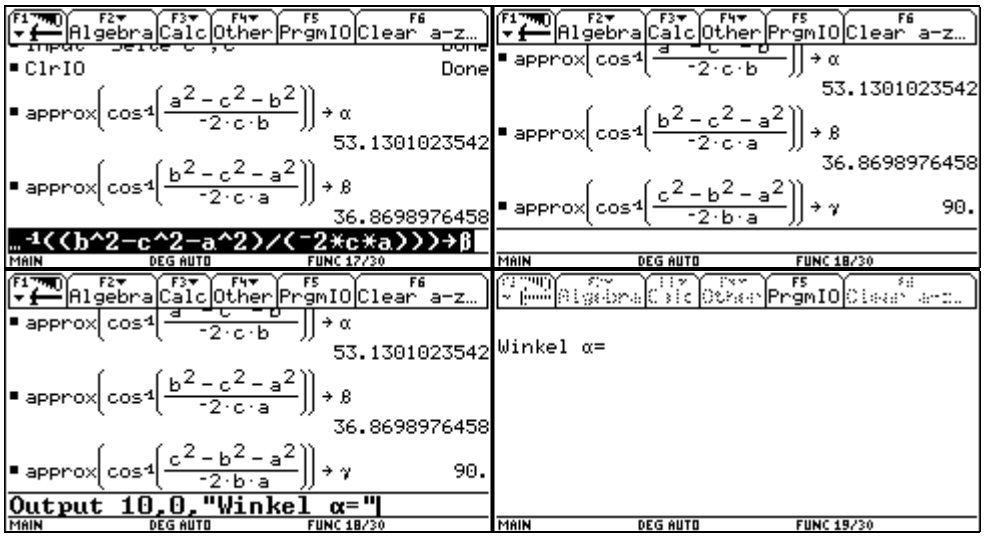




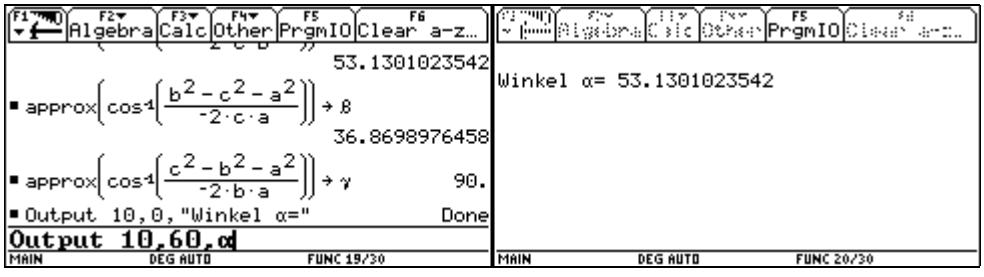
Danach wird der PrgIO wieder gelöscht.



Es folgen die Berechnungen der Winkel nach dem Cosinussatz. Die dritte Berechnung könnte auch über die Winkelsumme im Dreieck durchgeführt werden.

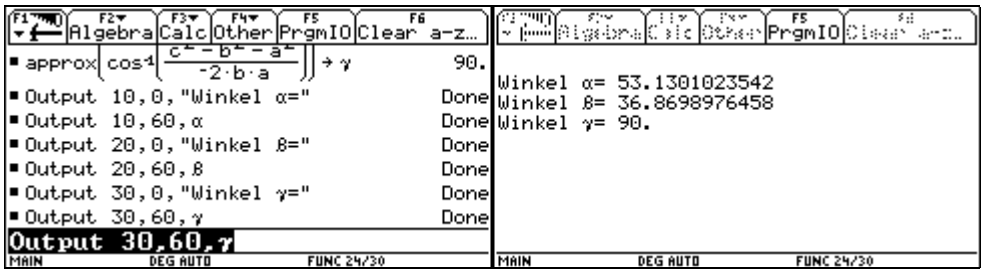


Mit dem Befehl Output kann sowohl ein Text als auch der Wert einer Variablen ausgegeben werden. Über die Angabe von Zeile,Spalte kann die Ausgabe am Bildschirm genau gesteuert werden. Der Bildschirm hat 102 Zeilen und 239 Spalten. Die Zeilen werden von oben und die Spalten von links gezählt. Ein Text muß unter immer unter " " gesetzt werden.

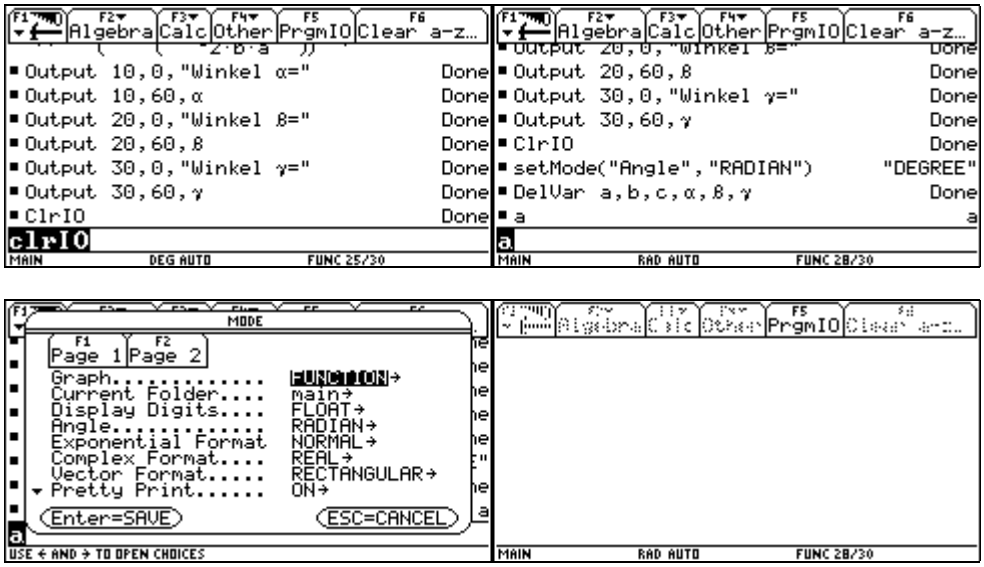


Eine Variable, deren Wert ausgegeben werden soll, muß ohne "angeführt werden. Alternativ könnte man zur Befehlsausgabe auch den Befehl **Disp** verwenden. In unserem Beispiel mit dem Befehl: Disp "Winkel α ="&string(α)

Dies ist einfacher, hat aber den Nach- oder Vorteil, dass mit jedem **Disp** Befehl eine neue Zeile ganz links begonnen wird.
Der **Output-Befehl** bietet also etwas mehr an Gestaltungsmöglichkeiten.



Abschließend wird der PrgIO gelöscht, der Berechnungsmodus für Winkel wieder auf RADIAN zurückgesetzt und alle vorkommenden Variablen gelöscht.



Lehrinhalte: Eingabe und Ausgabe über den Input-Output Schirm, Programmunterbrechung

Befehle: Input, Output, Disp, Pause, ClrIO, setMode, DelVar, approx

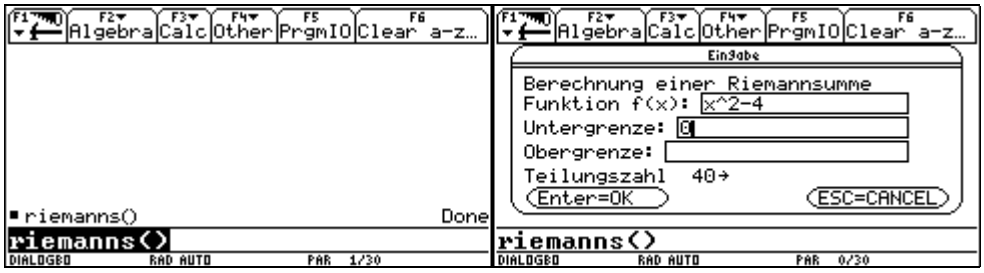
Übung: Berechnung eines schiefwinkeligen Dreiecks aus zwei Seiten und dem eingeschlossenen Winkel

2.3 Steuerung über den Input-Output-Schirm

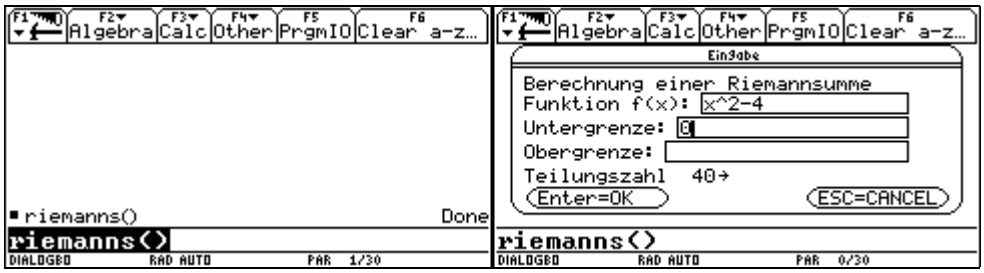
Programm: Berechnung einer Riemannsumme

Benutzung:

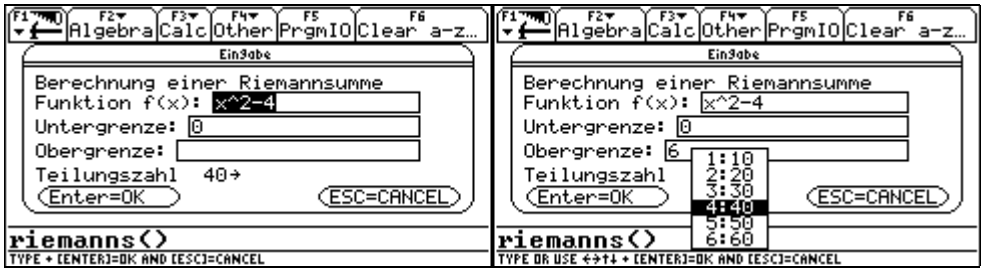
Nach dem Aufruf des Programms erscheint eine Dialogbox. Über [ESC] kann man das Programm sofort abbrechen.



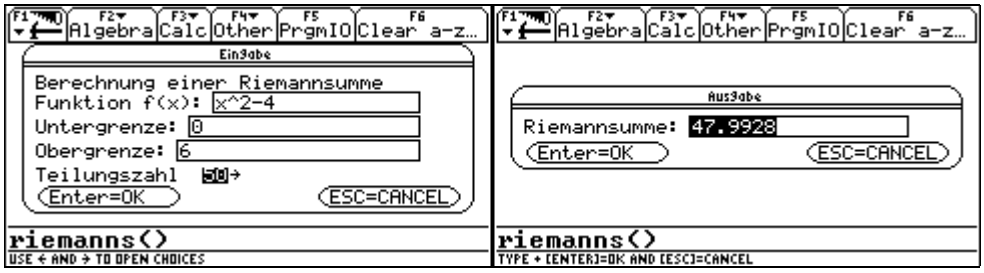
Wird in dieser Box [ENTER] gedrückt bevor alle Parameter eingegeben worden sind, dann erscheint die Box nochmals.

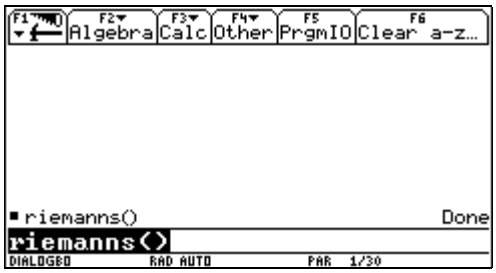


In der letzten Zeile der Box kann über ein **Drop-down**-Menü die Teilungszahl bestimmt werden.

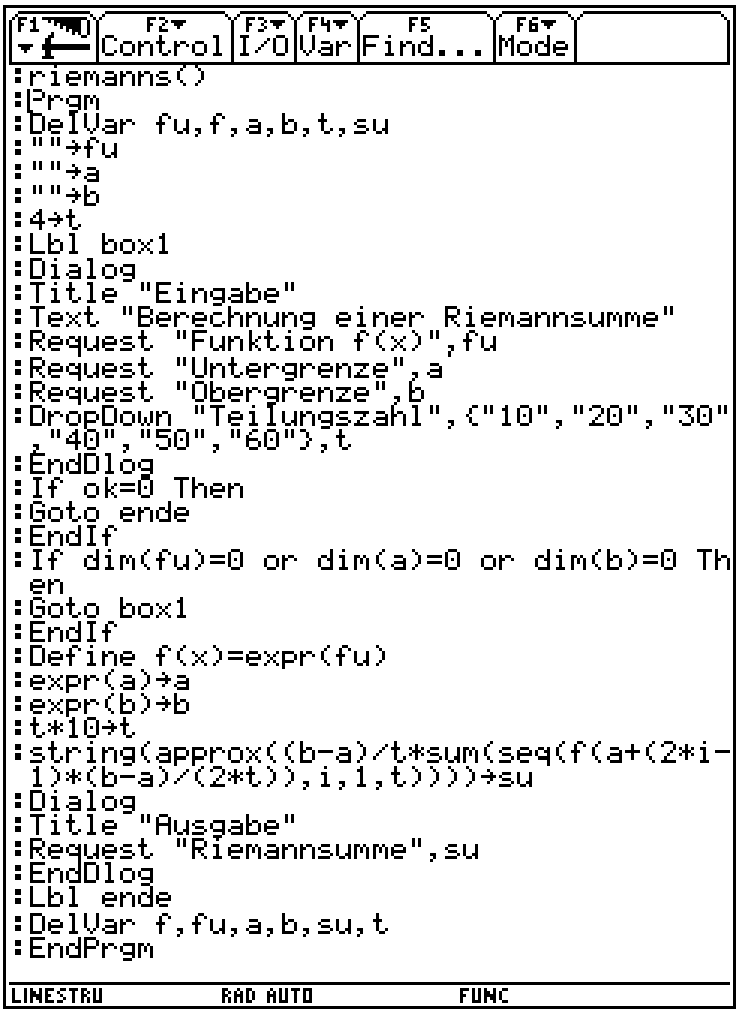


Wird die vollständig ausgefüllte Box mit [ENTER] verlassen, so erscheint eine weitere Box mit dem Ergebnis. Diese führt durch [ENTER] oder [ESC] zur Beendigung des Programms.



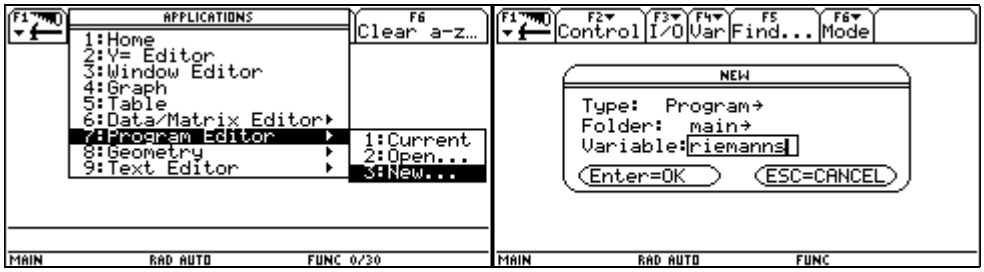


Programmcode:

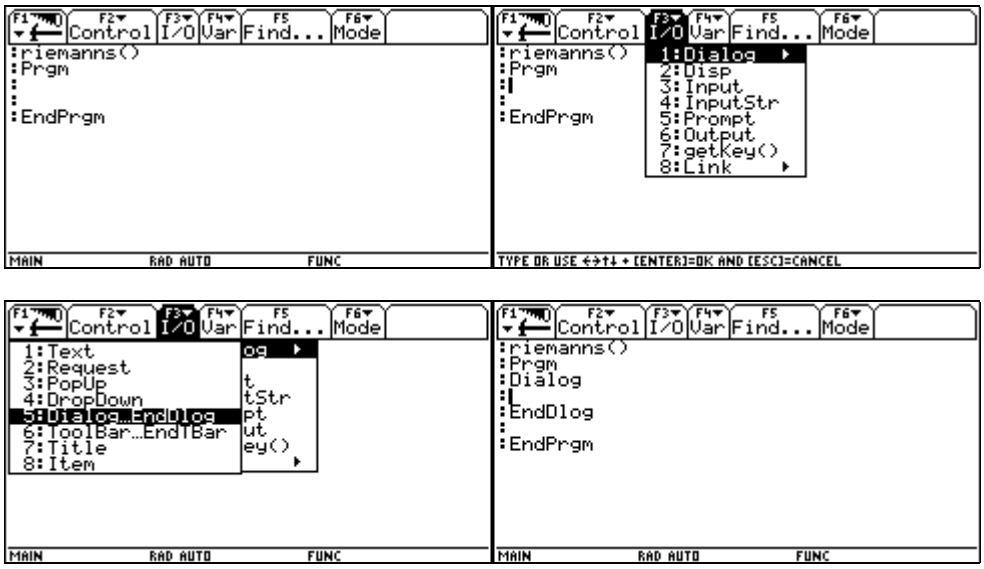


Erstellen des Programms:

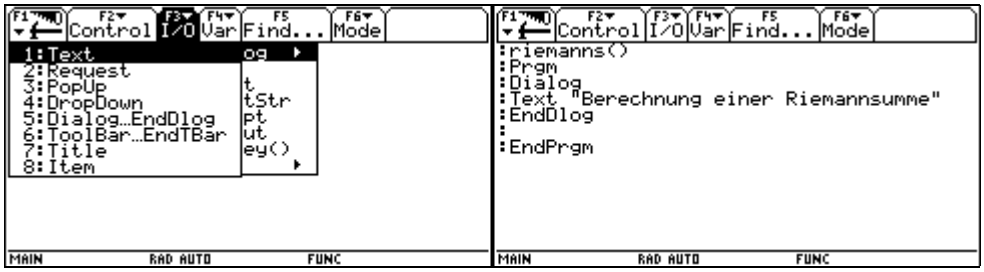
Der Programm Editor wird geöffnet und ein neues Programm mit dem Namen riemanns angelegt.

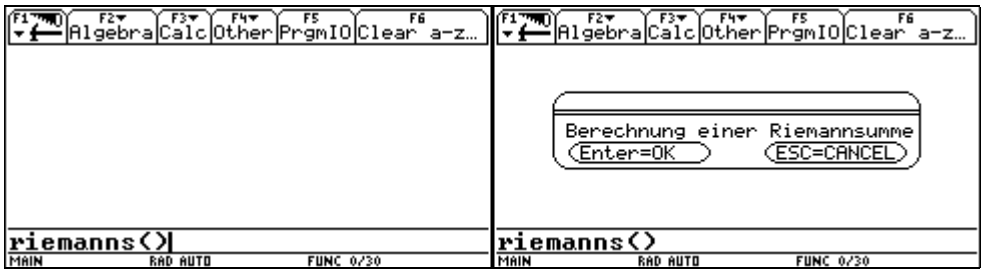


Anfang und Ende des Programms werden automatisch eingeblendet. Wir wenden uns zunächst dem Erstellen der Dialogbox für die Eingabe zu. Unter F3 finden wir bei Dialog das **Befehlspaar** Dialog - EndDlog. Damit sind Anfang und Ende der Befehlskette einer Dialogbox festgelegt.

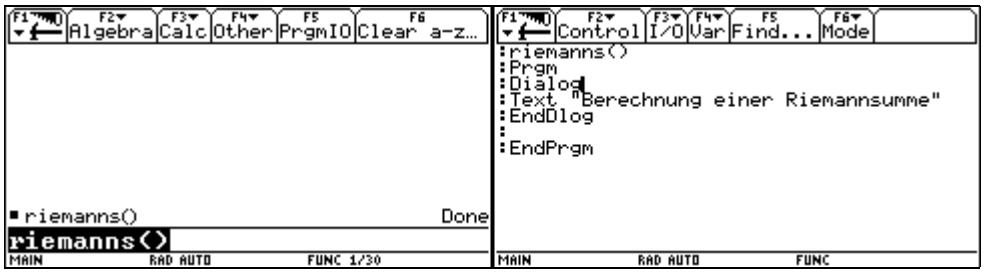


Ebenfalls unter F3 1:Dialog findet man den **Befehl** Text mit dem Text in die Box geschrieben werden kann. Wir schreiben gleich den entsprechenden Text, kehren mit 2nd ESC in den Homebereich zurück und testen unser Programm. Text wird in der Box nicht umgebrochen. Man muß eigenständig gegebenenfalls weitere Zeilen mit dem Befehl Text gestalten.

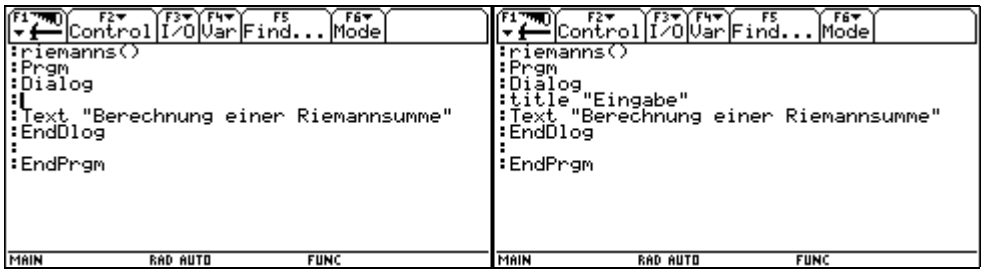




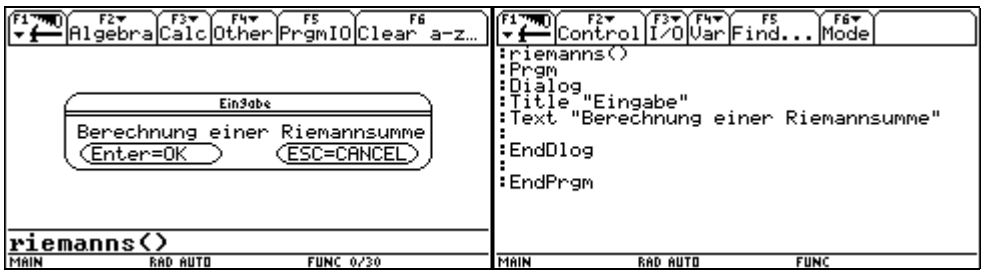
Mit APPS 7 und 1 gelangen wir in den Programmeditor zurück.



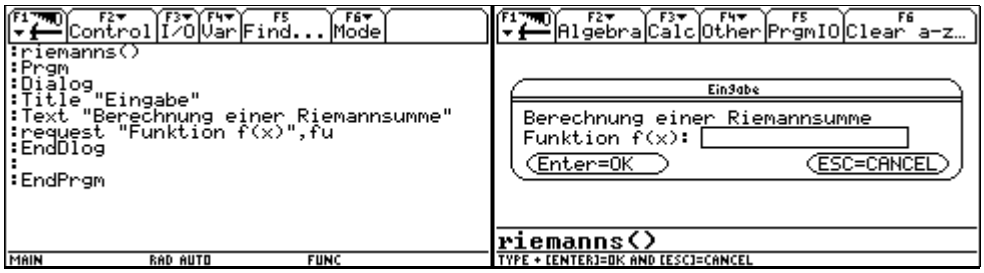
Wir können der Box auch eine Überschrift verleihen. Das wird mit dem Befehl **Title** erreicht.



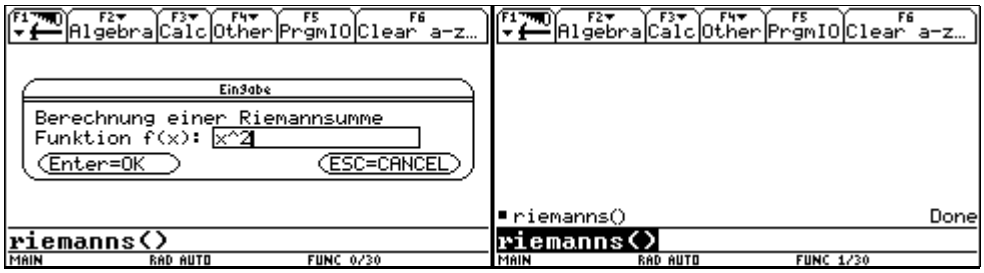
Auch diese Veränderung betrachten wir sofort mit einem Programmstart. Zurückgekehrt in den Editor verwenden wir den Befehl Request zur Eingabe von Werten durch den Benutzer. Dieser Befehl wird gefolgt von Text unter "" und getrennt durch einen Beistrich von einem Variablennamen, unter dem die Eingabe des Benützers als **String** gespeichert wird.



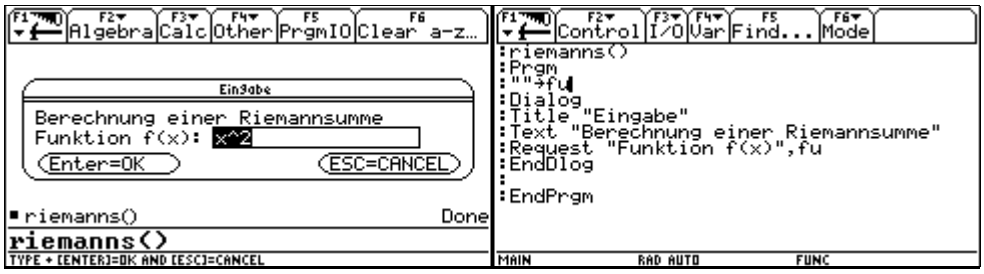
Auch diese Veränderung unserer Box wollen wir gleich durch Starten des Programms bewundern.



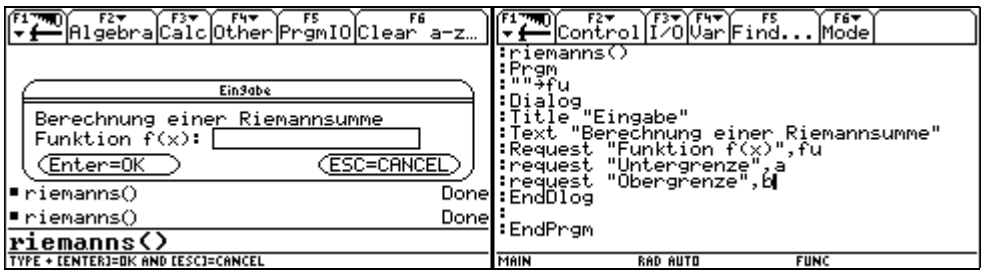
Wir geben auch einen Term für die Funktion ein.



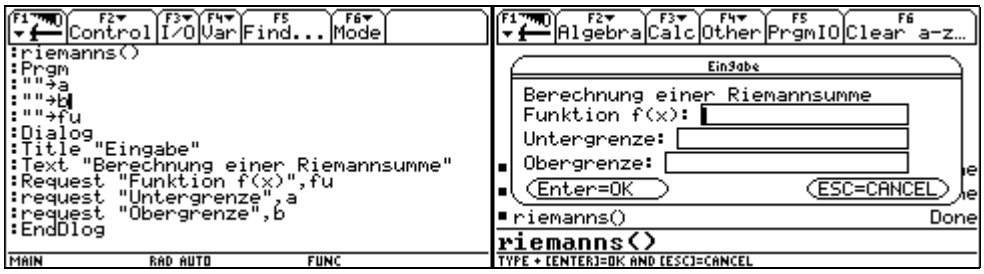
Wird das Programm noch einmal gestartet, so erscheint der eben eingegebene Term automatisch wieder. Um das zuverhindern können wir der Variablen fu vor dem Aufruf der Box einen leeren **String** "" zuweisen.



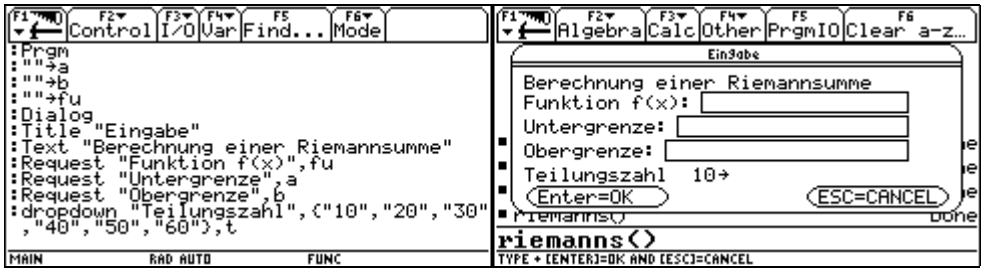
Wenn wir nun noch einmal starten, erscheint die Eingabezeile in der Box wieder leer.



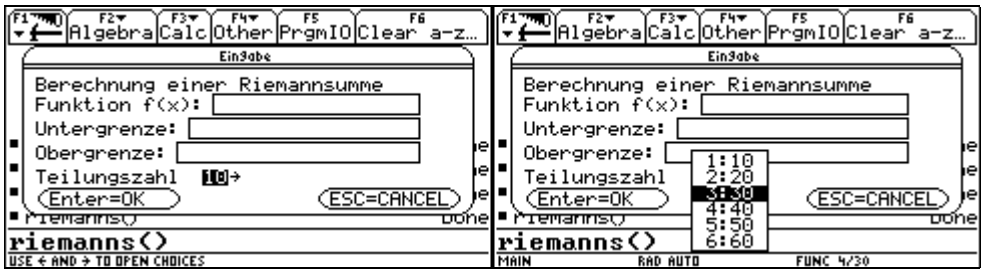
Mit weiteren Request Befehlen werden die Eingabeaufforderungen für Untergrenze und Obergrenze geschaffen. Auch die Variablen *a* und *b* werden vor dem Start der Box mit leeren Strings initialisiert. Diese Neuerungen können sofort überprüft werden.

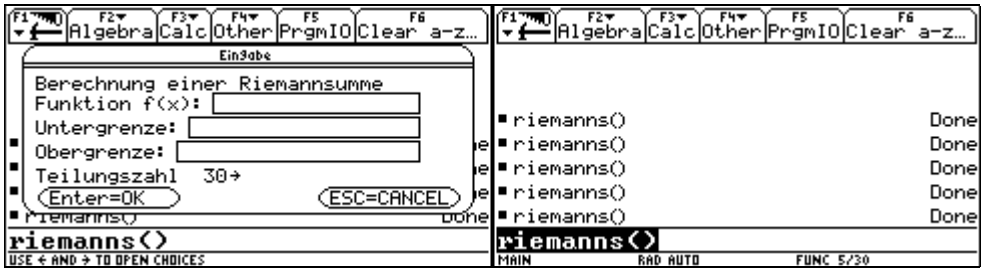


Jetzt wird das Drop-down-Menü erzeugt. Der Befehl DropDown wird gefolgt von einem **Text unter ""**. Nach einem Beistrich folgt eine **Liste** innerhalb von **geschwungenen Klammern** {}, in der sich die Texte für die Optionen befinden (durch Beistriche getrennt) zwischen denen gewählt werden kann. Nach einem weiteren Beistrich folgt ein Variable für die gewählte Option. Wählt der Benutzer zum Beispiel den 3. Text der Liste, so hat diese Variable nach Verlassen der Box den Wert 3. Auch diese Veränderung betrachten wir nach einem Programmstart.

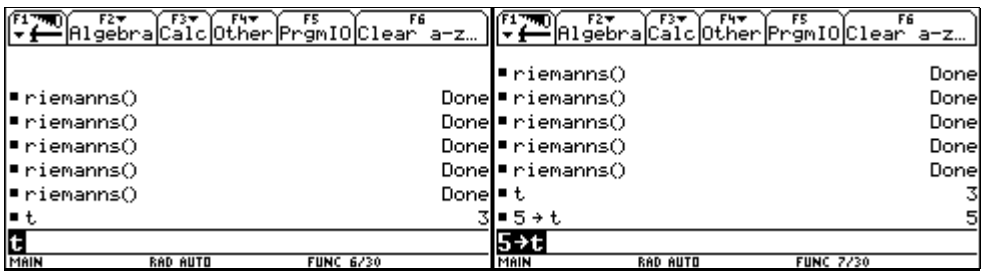


Wir wählen als Teilungszahl 30 und beenden das Programm.

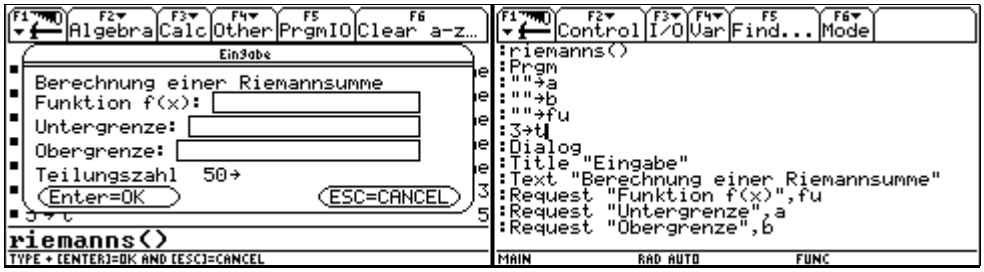




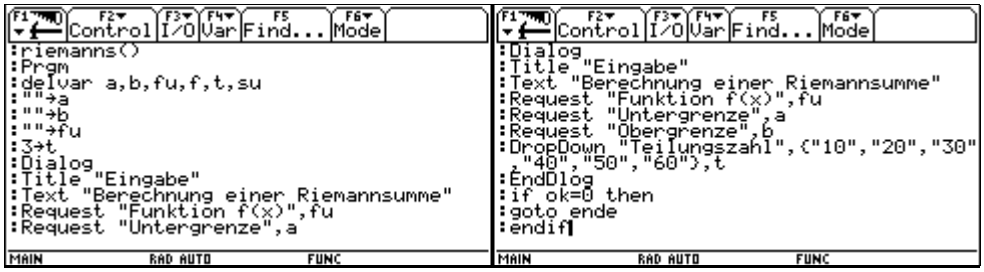
Wie angekündigt besitzt die Variable *t* den Wert 3. Nun weisen wir ihr den Wert 5 zu.



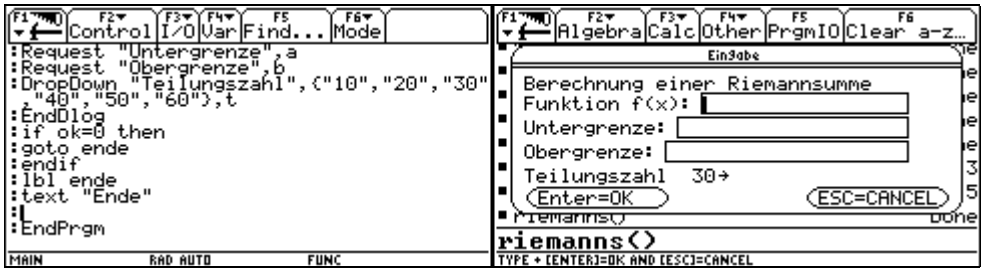
Bei neuerlichem Programmstart hat die Teilungszahl nun den Wert 50. Der voreingestellte Wert läßt sich also über die Belegung von *t* steuern. Wir wählen in unserem Programm z. B. für *t* den Wert 3, also 30 als voreingestellte Teilungszahl.



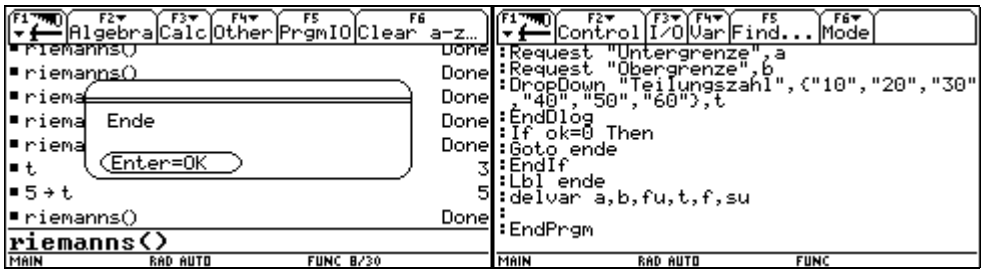
Nun soll festgelegt werden, was geschehen soll, wenn ein Benützer in der Box ESC wählt. Ob in der Box ESC gedrückt wurde oder nicht, wird von der **Systemvariablen** ok wahrgenommen. Besitzt sie den Wert 0, so wurde ESC gedrückt. Wir springen in diesem Fall mit einem **Sprungbefehl Goto** zu einer **Markierung** Lbl **(genannt Label)** am Ende des Programms. Diesem Label geben wir die Bezeichnung ende. Um diese Veränderung testen zu könne, geben wird nach dem Label ende noch einen Textbefehl ein.



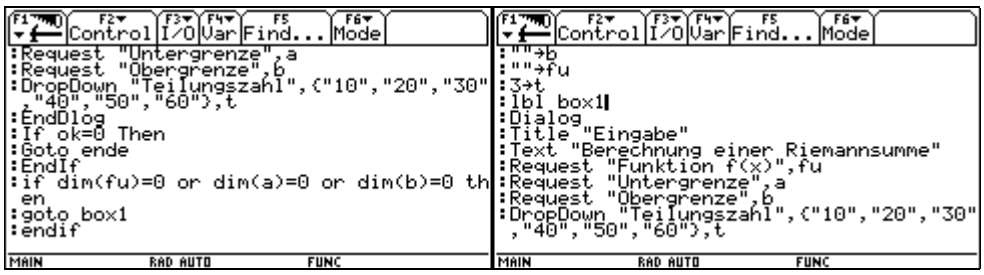
Wir starten das Programm wieder und verlassen die Box mit ESC.



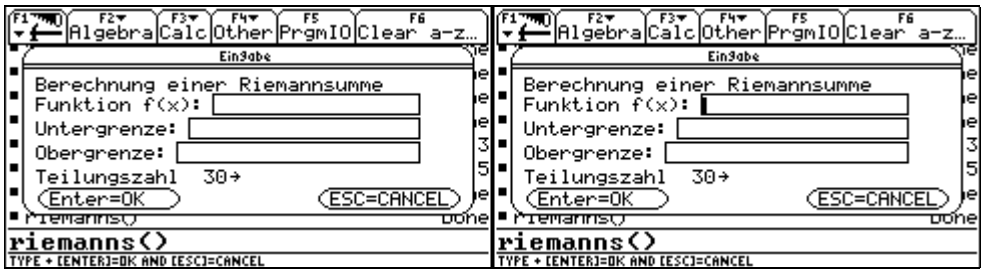
Die Box mit dem Text "Ende" bestätigt die Richtigkeit unserer Programmierung. Nach Lbl ende führen wir das Löschen der Variablen durch.



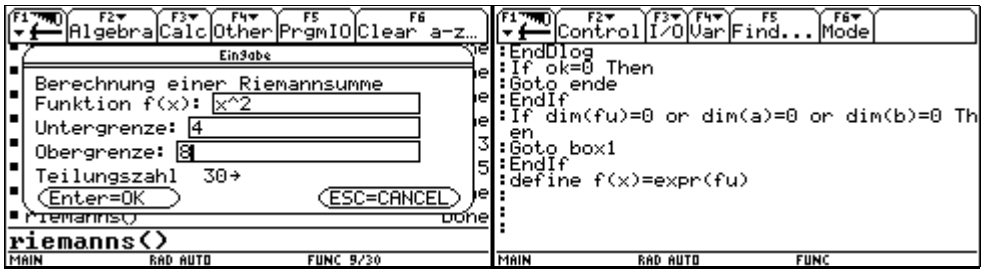
Jetzt wollen wir noch verhindern, dass die Box mit ENTER verlassen wird, wenn auf eine Eingabe vergessen wurde. Dazu untersuchen wir die Länge der eingegebenen Strings mit der Funktion **dim**. Ist die Länge Null, dann fehlt die entsprechende Eingabe. Wir setzen für diesen Fall einen Sprungbefehl zu einem Label box1, das sich direkt vor dem Beginn der Dialogbox befindet.



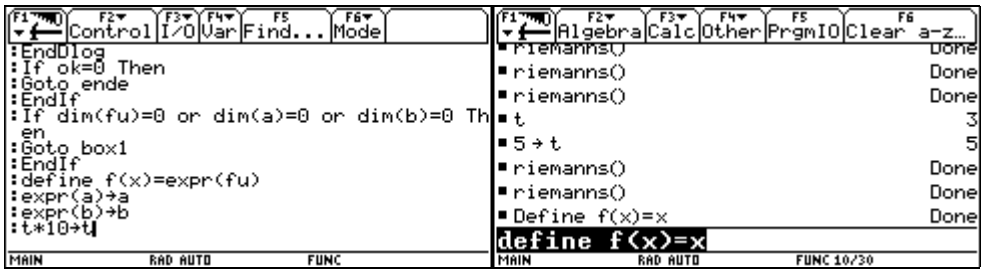
Auch diese Neuerung untersuchen wir sofort. Tatsächlich erscheint die Box solange wieder bis alle Eingaben durchgeführt wurden.



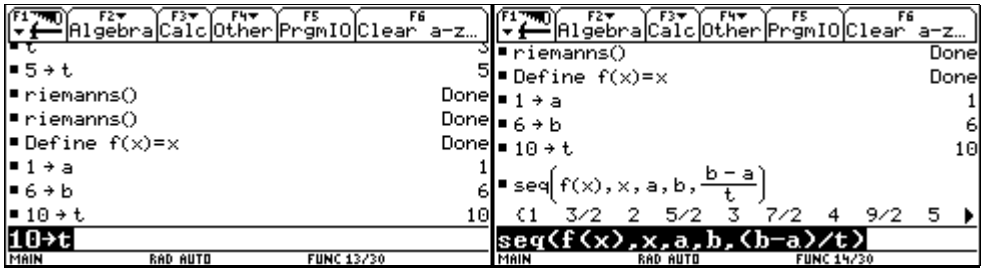
Über die Funktion expr wird der in fu als String gespeicherte Funktionsterm in einen Term verwandelt und anschließend mit Define als eine Funktion f(x) festgelegt.



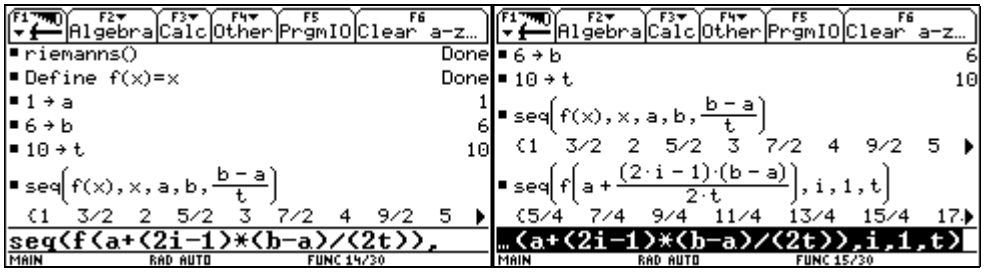
Auch die für untere und obere Grenze in *a* und *b* abgespeicherten Strings werden mit expr in Zahlenwerte verwandelt. Aus dem Wert von *t* wird die Teilungszahl 10*t* errechnet. Um die Berechnung der Riemannsumme besser verstehen zu können, definieren wir im Homebereich die identische Funktion als f(x).



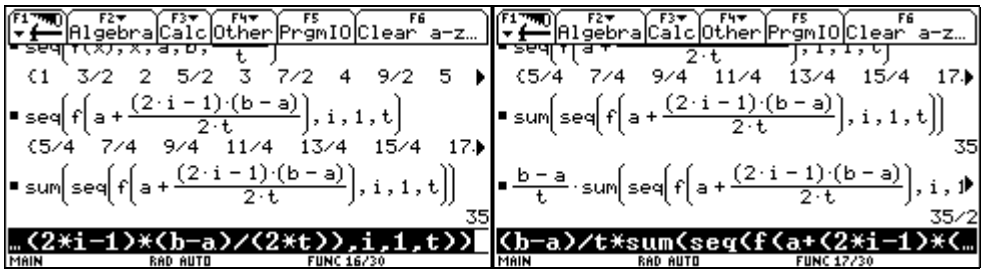
Der Befehl seq erzeugt eine Folge von Funktionswerten von $f(x)$ (in unserem Fall $f(x) = x$) für x von a bis b mit der Schrittweite $(b-a)/t$. Für die gewählten Werte für a, b und t ergibt sich folgende Liste an Teilungspunkten des Intervalles $[a,b]$ bei t gleichen Teile.

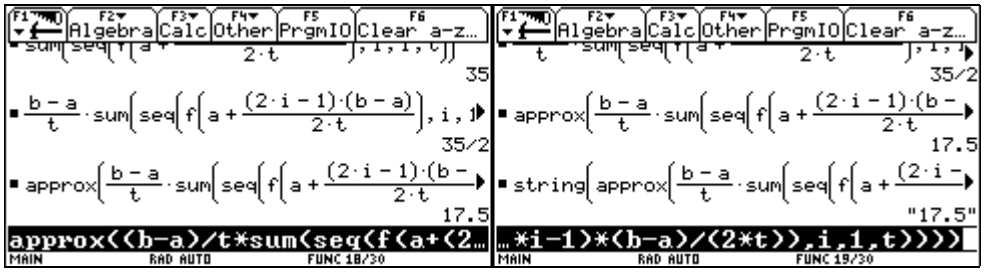


Wollen wir für die Riemannsumme die Funktionswerte in der Mitte der Teilintervalle verwenden, dann müssen wir die Erzeugung der Folge abändern:

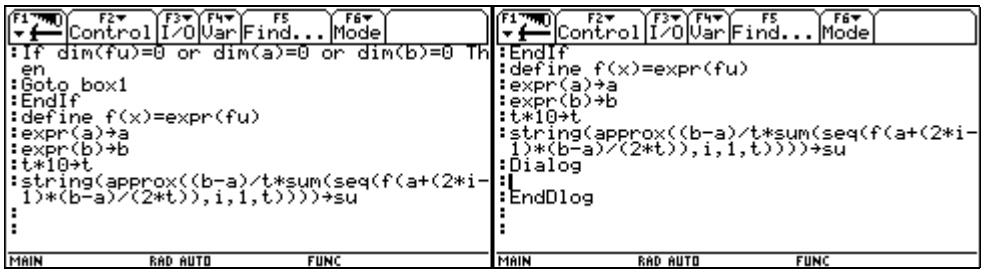


Mit **sum** wird die Liste der Funktionswerte addiert und muß nur noch mit der konstanten Teilintervalllänge $(a-b)/t$ multipliziert werden, um die Riemannsumme zu ergeben. Mit approx lassen wir uns einen Näherungswert errechnen und verwandeln diesen mit **string** in eine Textkonstante. Diesen – nun schon recht langen - Befehl kopieren wir mit ♦ C in die Zwischenablage.

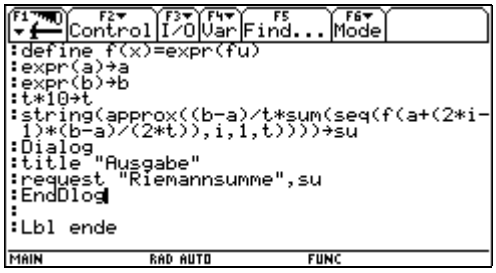




Nach dem wir unser Programm wieder geöffnet haben fügen wir den Befehl zur Berechnung der Riemannsummen mit ♦ V entsprechend ein. Die Riemannsumme wird als **String** der Variablen su zugewiesen, die in der folgenden Dialogbox über den **Request-Befehl** ausgegeben wird.



Da das Programm nach Ausgabe dieser Box beendet werden soll, brauchen wir für das Drücken von ENTER bzw. ESC keine Vorkehrungen mehr zu treffen.



Lehrinhalte: Erstellen von Dialogboxen, Eingabe und Ausgabe über Dialogboxen, Programmierung von ENTER und ESC in den Boxen, die Systemvariable ok, Sprungbefehl und Sprungmarke.

Befehle: DelVar, Text, Request, DropDown, Title, Dialog-EndDlog, Goto, Lbl, dim, expr, sum, seq, If-Then-EndIf, Define, string, approx.

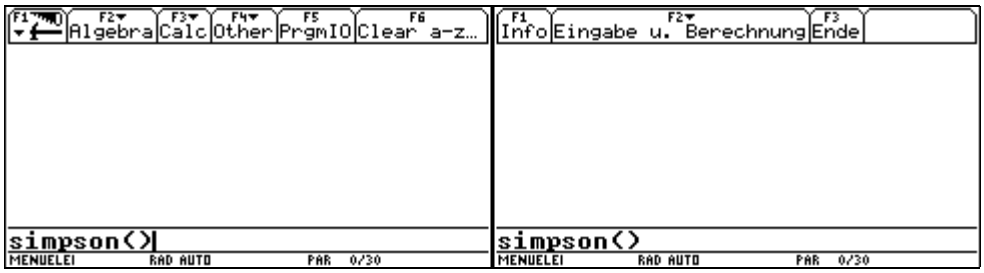
Üung: Berechnung eines Rotationsvolumens

2.4 Steuerung über eine Menüleiste

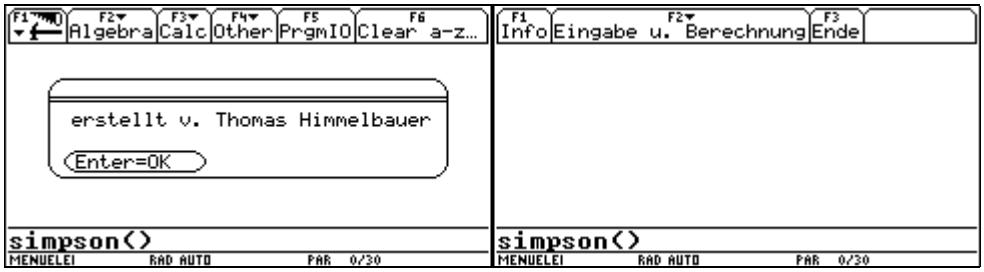
Programm: Nummerische Integration nach Simpson

Benutzung:

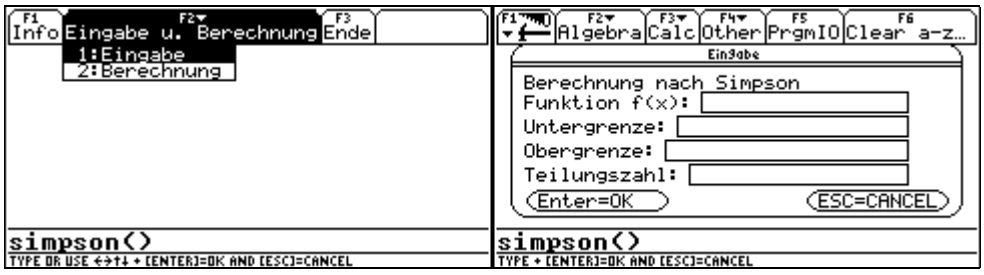
Wir starten das Programm und es erscheint eine neue Menüleiste.



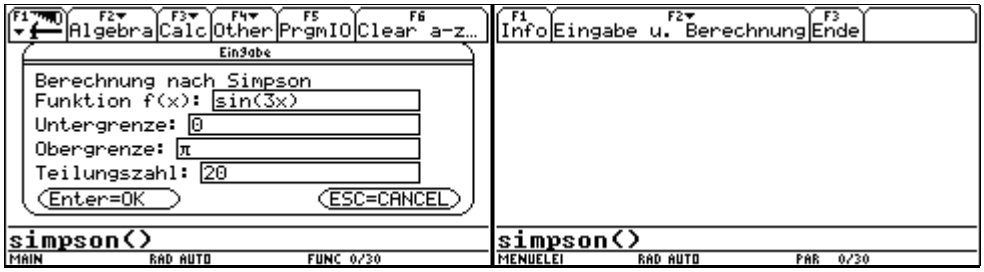
Unter F1 finden sich Informationen.



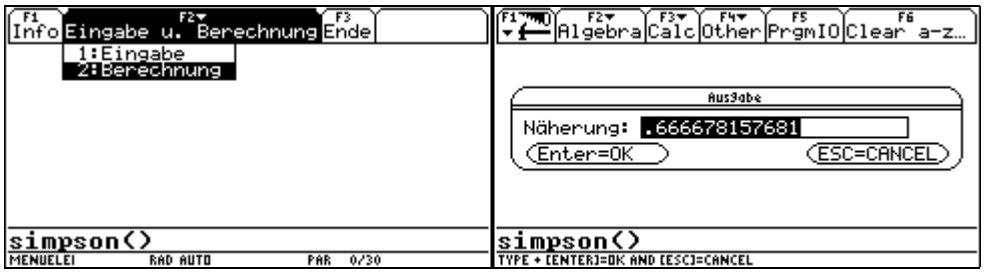
F2 führt zu den Optionen Eingabe und Berechnung. Wir wählen 1:Eingabe und können die nötigen Daten über eine Dialogbox eingeben.



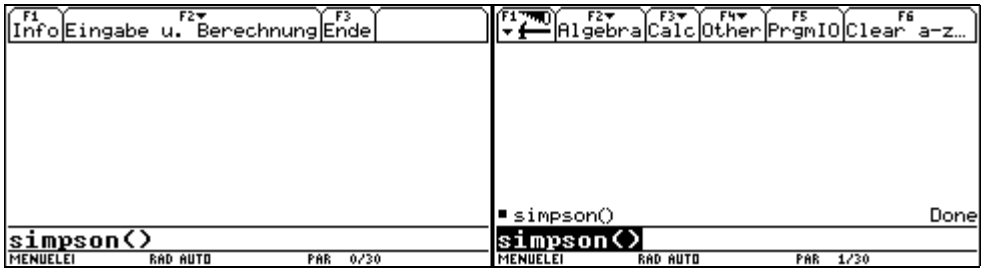
Nach dem Verlassen der Box kehrt man in das Menü zurück.



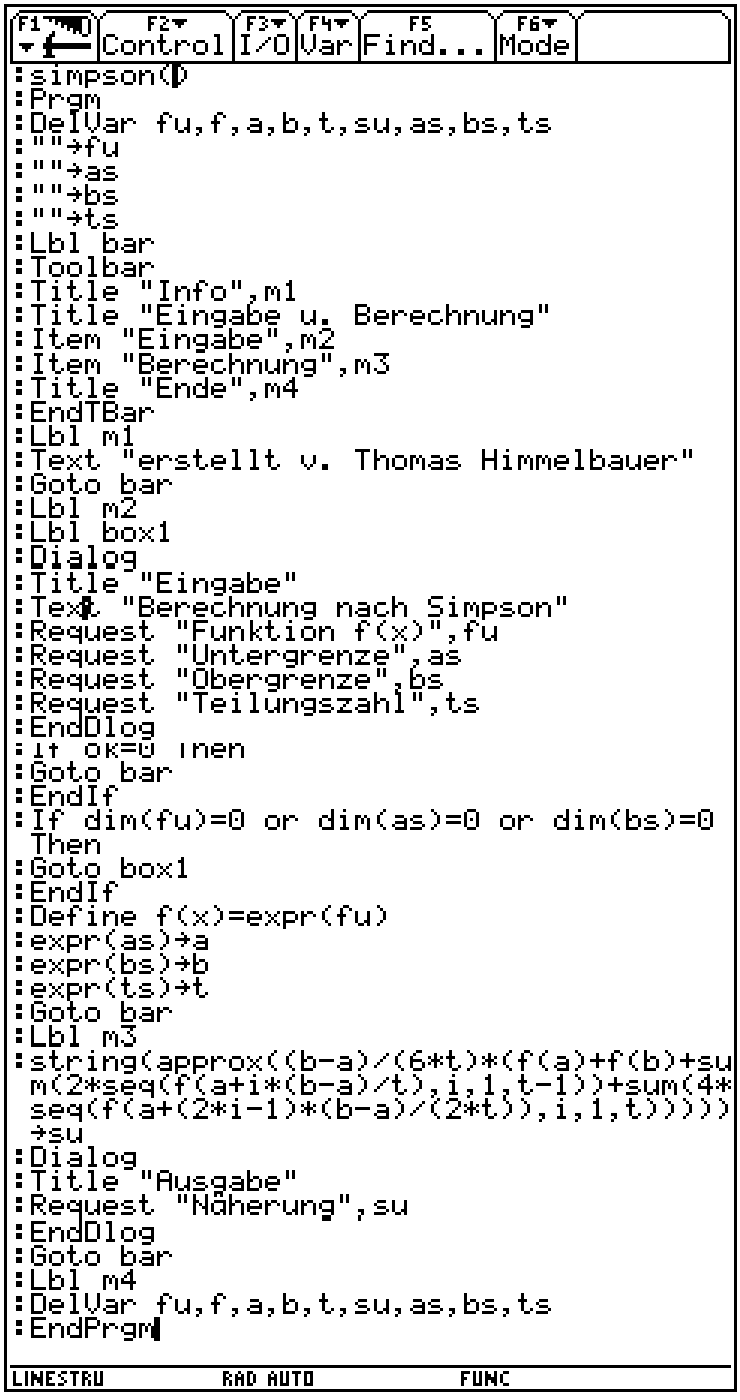
Mit F2 2:Berechnung gelangen wir zum Ergebnis.



Nach Verlassen der Box gelangen wir wieder ins Menü und können mit F3 das Programm ordnungsgemäß beenden.



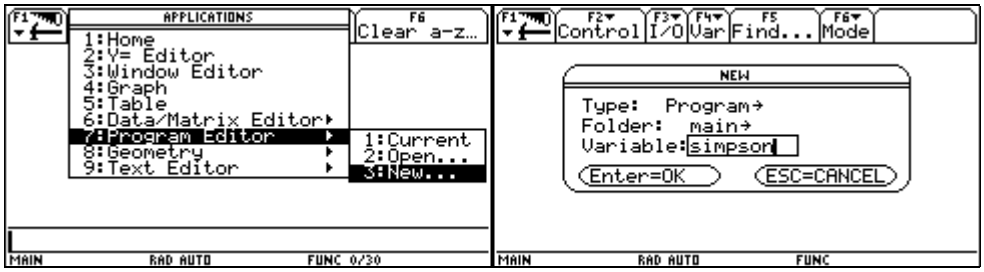
Programmcode:



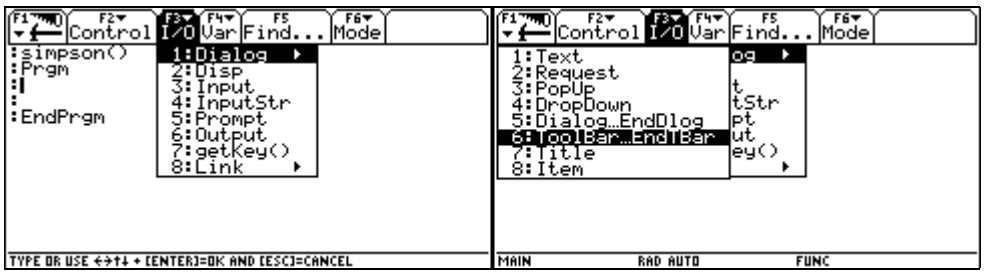
```
F1 F2 Control F3 I/O F4 Var F5 Find... F6 Mode
:simpson()
:Prgm
:DelVar fu,f,a,b,t,su,as,bs,ts
:""→fu
:""→as
:""→bs
:""→ts
:Lbl bar
:Toolbar
:Title "Info",m1
:Title "Eingabe u. Berechnung"
:Item "Eingabe",m2
:Item "Berechnung",m3
:Title "Ende",m4
:EndTBar
:Lbl m1
:Text "erstellt v. Thomas Himmelbauer"
:Goto bar
:Lbl m2
:Lbl box1
:Dialog
:Title "Eingabe"
:Text "Berechnung nach Simpson"
:Request "Funktion f(x)",fu
:Request "Untergrenze",as
:Request "Obergrenze",bs
:Request "Teilungszahl",ts
:EndDlog
:If ok=0 Then
:Goto bar
:EndIf
:If dim(fu)=0 or dim(as)=0 or dim(bs)=0
 Then
:Goto box1
:EndIf
:Define f(x)=expr(fu)
:expr(as)→a
:expr(bs)→b
:expr(ts)→t
:Goto bar
:Lbl m3
:string(approx((b-a)/(6*t)*(f(a)+f(b)+su
 m(2*seq(f(a+i*(b-a)/t),i,1,t-1))+sum(4*
 seq(f(a+(2*i-1)*(b-a)/(2*t)),i,1,t)))))
 →su
:Dialog
:Title "Ausgabe"
:Request "Näherung",su
:EndDlog
:Goto bar
:Lbl m4
:DelVar fu,f,a,b,t,su,as,bs,ts
:EndPrgm
LINESTRU RAD AUTO FUNC
```

Erstellung der Programms:

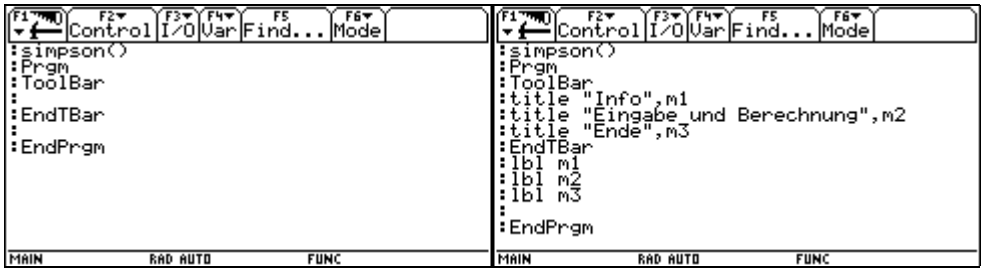
Wir öffnen den Programmeditor und legen eine neues Programm mit dem Namen simpson an.



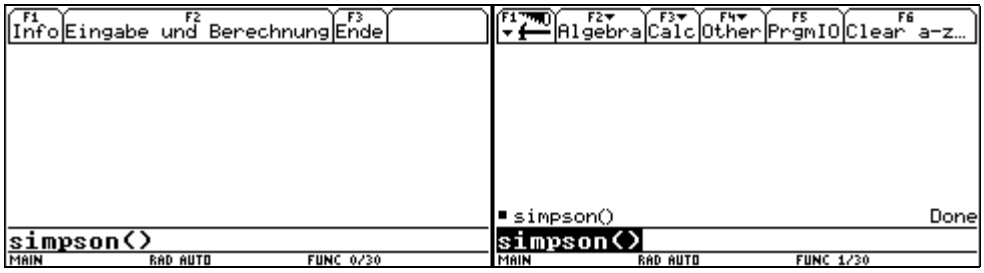
In F3 können wir unter Dialog das Befehlspaar ToolBar-EndTBar finden, das für die Schaffung einer programmeigenen Menüleiste zuständig ist.



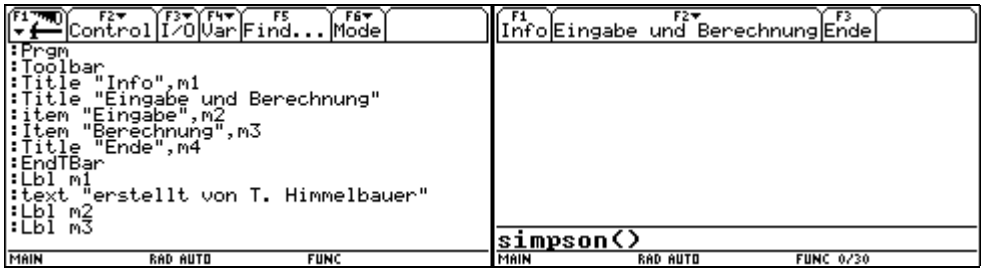
Mit den Befehl Title werden Texte als Bezeichnungen für die Funktionstasten F1,F2,F3... festgelegt. Der erste Befehl Title legt den Namen für F1, der zweite für F2 fest und so fort. Von den Texten durch einen Beistrich getrennt, folgen die Bezeichnung der **Label**, zu denen im Programm gesprungen wird, wenn die jeweilige Funktionstaste gewählt wird. Daher setzen wir hinter dem Befehl EndTBar die entsprechenden Label = Sprungmarken.



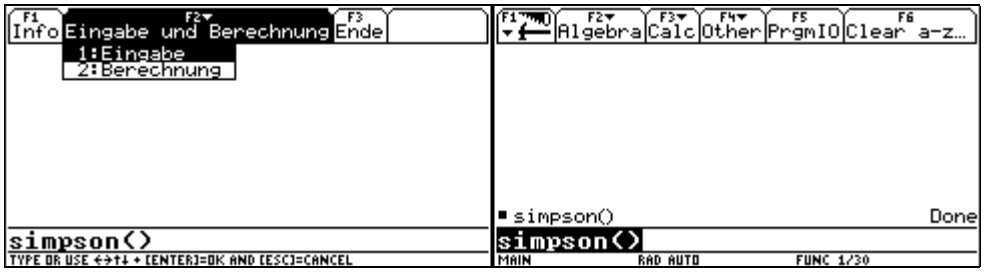
Nun probieren wir das Programm sofort aus. Tatsächlich erscheint die gewünschte Menüleiste. Bei Betätigung irgendeiner Funktionstaste, wird das Programm verlassen.



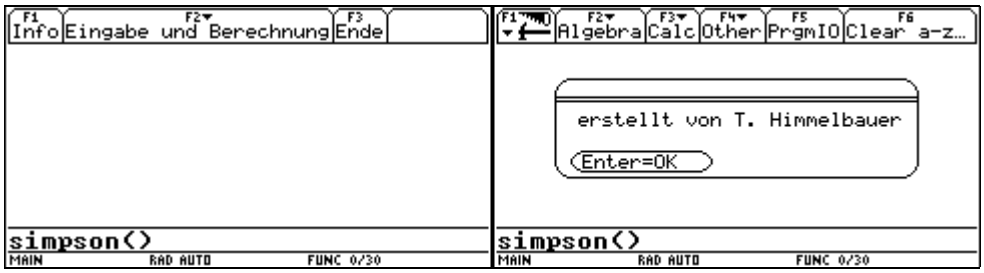
Mit dem Befehl **Item** können Unterpunkte für die Funktionstasten geschaffen werden. Für F2 legen wir die beiden Unterpunkte Eingabe und Berechnung an. Wenn Unterpunkte vorhanden sind, so sind für diese und nicht für die Hauptüberschrift Label anzulegen. Außerdem setzen wir hinter das Label m1, der für den Menüpunkt F1 Info steht, einen kurzen Informationstext. Die veränderte Menüleiste wird sofort getestet.



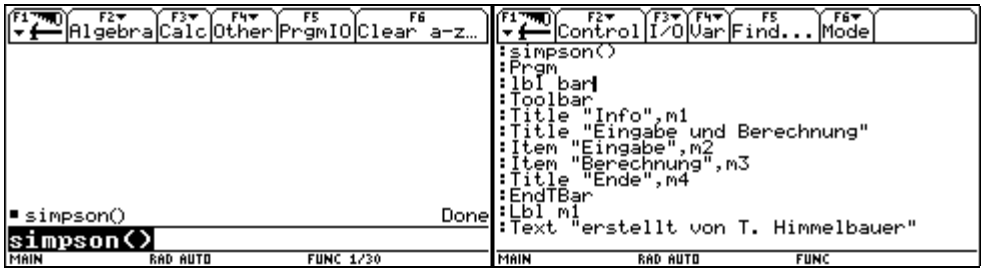
Bei F2 gibt es nun die beiden Unterpunkte. Wenn wir einen auswählen, wird das Programm beendet, da die entsprechenden Prozeduren noch fehlen



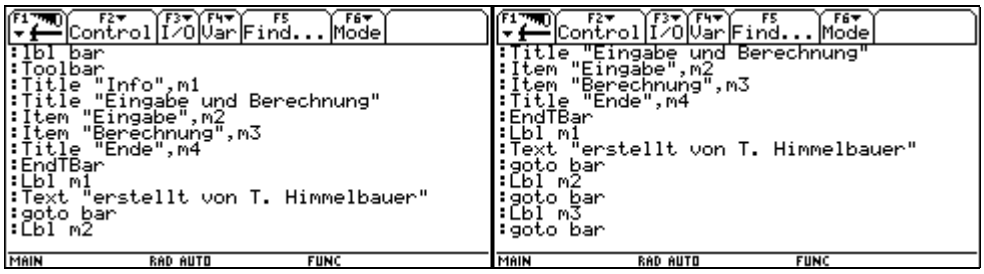
Wir starten wieder und wählen F1. Es erscheint eine Box mit unserem Text.



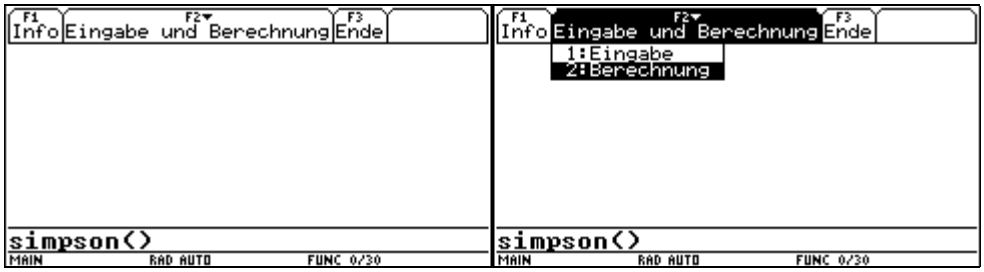
Nach ENTER wird das Programm beendet, ohne zur Menüleiste zurückzukehren. Daher kehren wir in das Programm zurück und setzen vor dem Befehl Toolbar das Label bar.

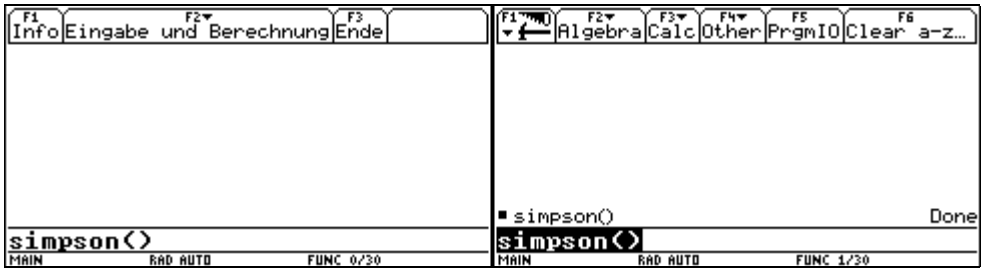


Dann setzen wir am Ende, der durch die Label m1, m2 und m3 bestimmten Abschnitte jeweils einen **Sprungbefehl** zurück zum Lbl bar. Damit wird immer wieder die Menüleiste aufgerufen. Nur beim Lbl m4, das ja bei der Auswahl von F3 Ende angesprungen wird, setzen wir keinen Sprungbefehl.

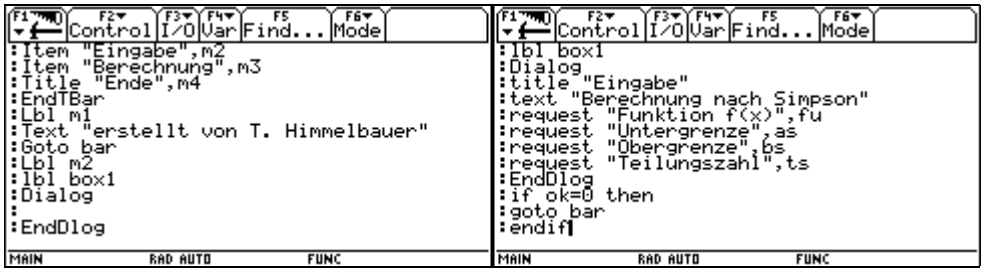


Auch diese Veränderung testen wir durch Starten des Programms. Nun erscheint die Menüleiste nach Auswahl eines Punktes immer wieder nur bei Auswahl von Ende wird das Programm beendet.

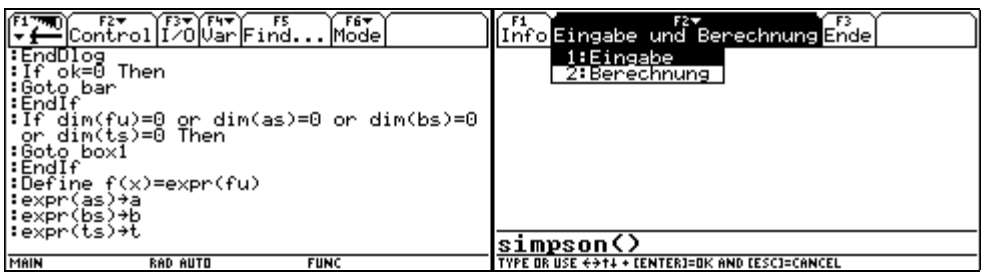




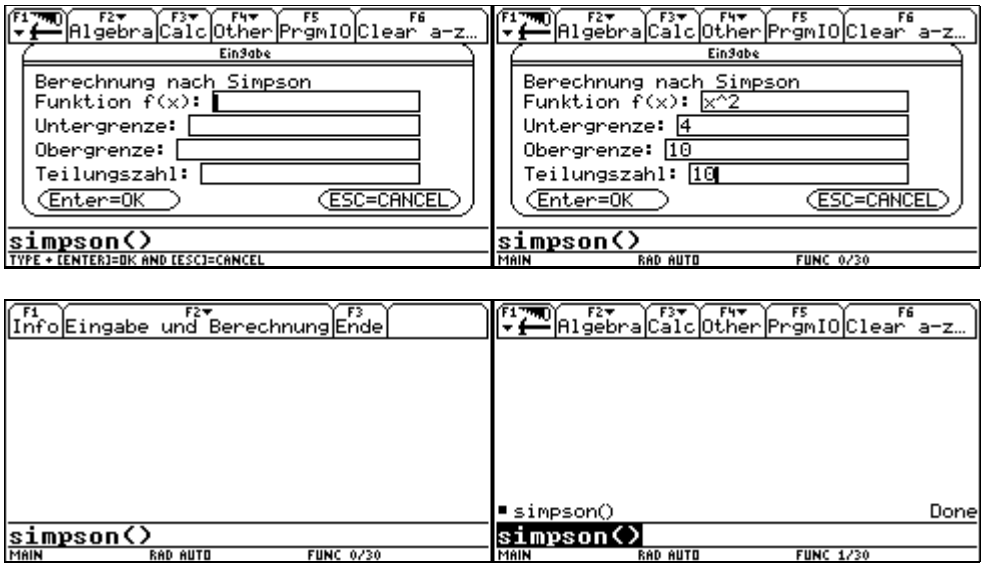
Für die Option Eingabe programmieren wir eine Dialogbox mit den im Abschnitt über Dialogboxen gelernten Befehlen und Anweisungen.



Die eingegebenen **Strings** verwandeln wir mit expr in mathematische Ausdrücke. Mit **Define** wird die Funktion $f(x)$ definiert. Dann probieren wir unsere Verbesserungen gleich aus. Wir wählen im Menü F2 Eingabe.

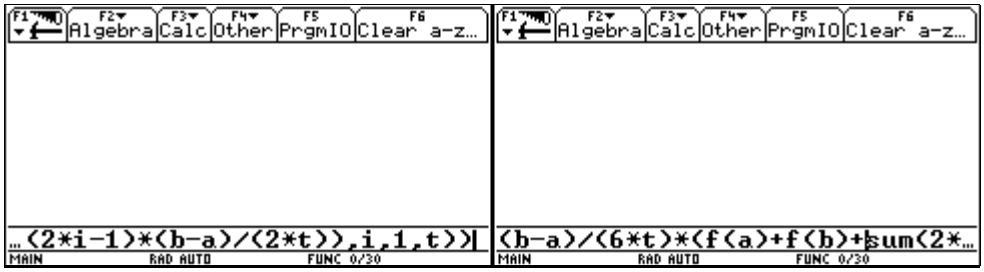


Dann legen wir die geforderten Werte fest. Mit ENTER verlassen wir die Box und kehren ins Menü zurück, wo wir das Programm mit F3 beenden.

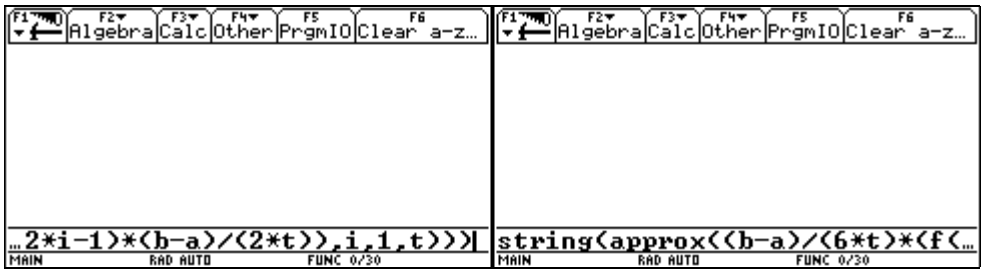


Es folgt die Überlegung zur Berechnung der Simpsonschen Näherungsformel. Dazu gehen wir in den Homebereich. Das Intervall [*a*,*b*] wird in *t* gleiche Teilintervalle geteilt. Dabei entstehen *t*-1 Teilungspunkte innerhalb des Intervalls [*a*,*b*]. Die doppelte Summe der Funktionswerte an diesen Teilungspunkten ist zu bilden. Außerdem ist die vierfache Summe der Funktionswerte an den *t* Halbierungspunkten der *t* Teilintervalle zu berechnen und diese beiden Teilsummen sind zu addieren.

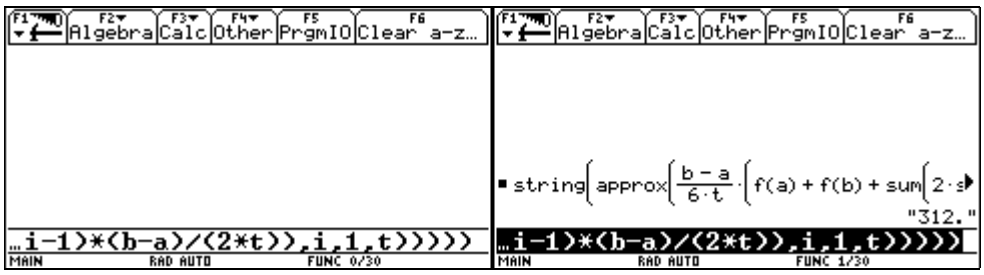
Der Funktionswert an den Stellen *a* und *b* ist dazuzuzählen und die entstandene Summe ist dann mit einem Sechstel der Länge eines Teilintervalles zu multiplizieren.



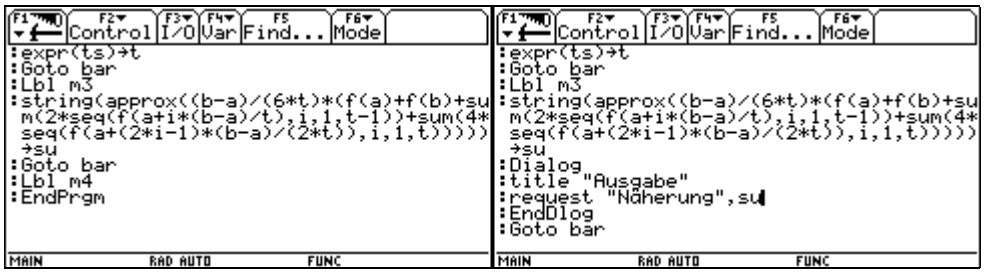
Mit approx erzwingen wir eine Dezimaldarstellung und mit string verwandeln wir das Ergebnis in einen String.



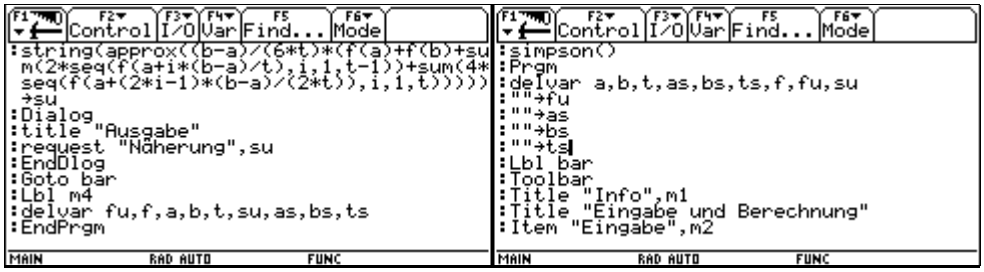
Mit ♦ C kopieren wir den Befehl in den Zwischenspeicher und ....



.... mit ♦V setzen wir die Näherungsformel an der entsprechende Stelle in unser Programm ein und weisen sie der Variablen su zu. Dann geben wir ihren Wert über eine Dialogbox mit Hilfe von Request aus.



Am Programmende und -anfang löschen wir alle Variablen. Außerdem initialisieren wir die Variablen der Eingabedialogbox mit leeren Strings.



Lehrinhalte: Erstellen einer Menüleiste, Kontrollvariable

Befehle: DelVar, Goto, Text, Request, Dialog-EndDlog, sum, seq, Toolbar-EndTBar, Item, Title, Lbl, Define, dim, expr, string, approx

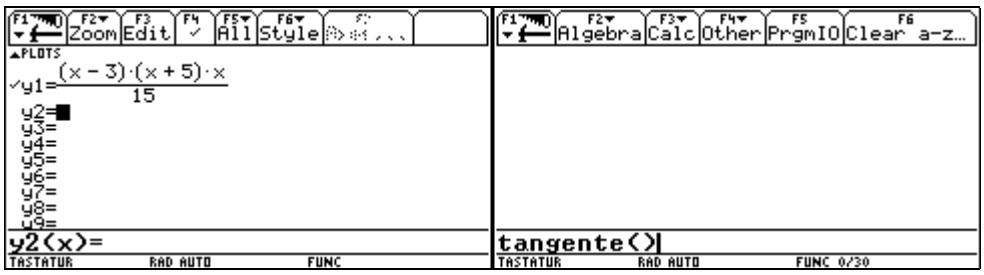
Übung: Nullstellenbestimmung nach Newton

2.5 Steuerung über Tasten

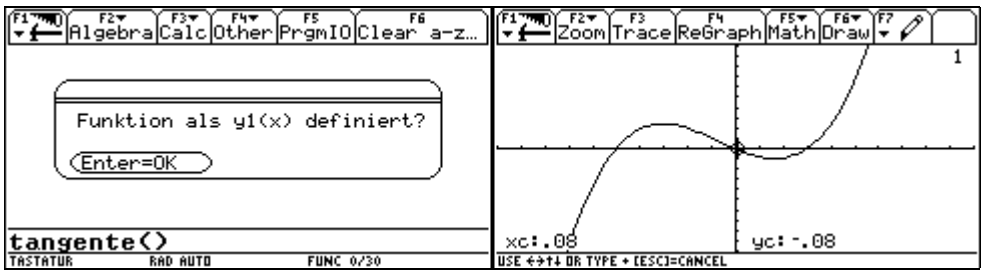
Programm: Bewegung einer Tangente entlang eines Graphen

Benutzung:

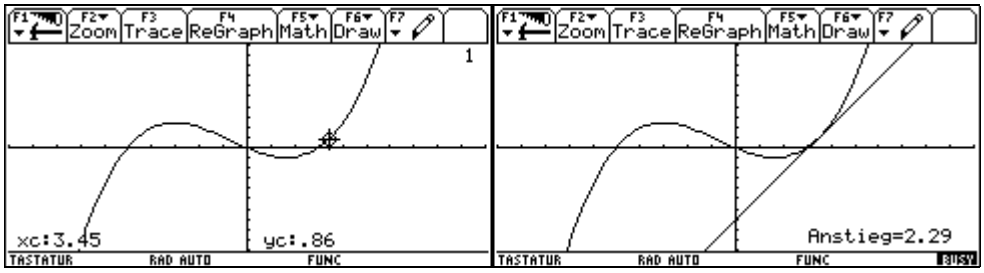
Wir geben den Funktionsterm, an den die Tangente in unterschiedlichen Punkten gezeichnet werden soll in den [Y=]Editor als y1(x) ein. Dann starten wir das Programm.



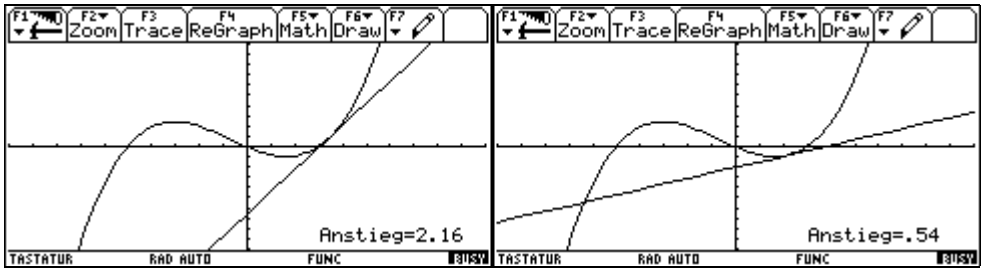
Es erscheint die Frage, ob man die Funktion auch als y1 abgespeichert hat.
Danach erscheint der Graph mit dem blinkenden Graphikcursor am Graphen. Wir können wie unter [F3] Trace den Graphen entlang fahren.



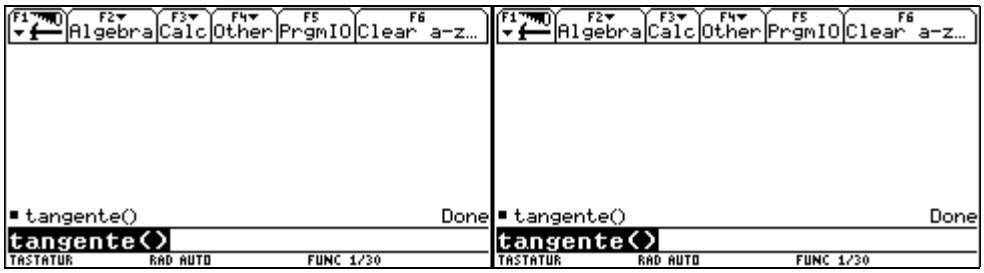
Die Stelle, an der die 1. Tangente gezeichnet werden soll, legen wir mit [ENTER] fest.
Die Tangente wird gezeichnet und ihr Anstieg angezeigt.



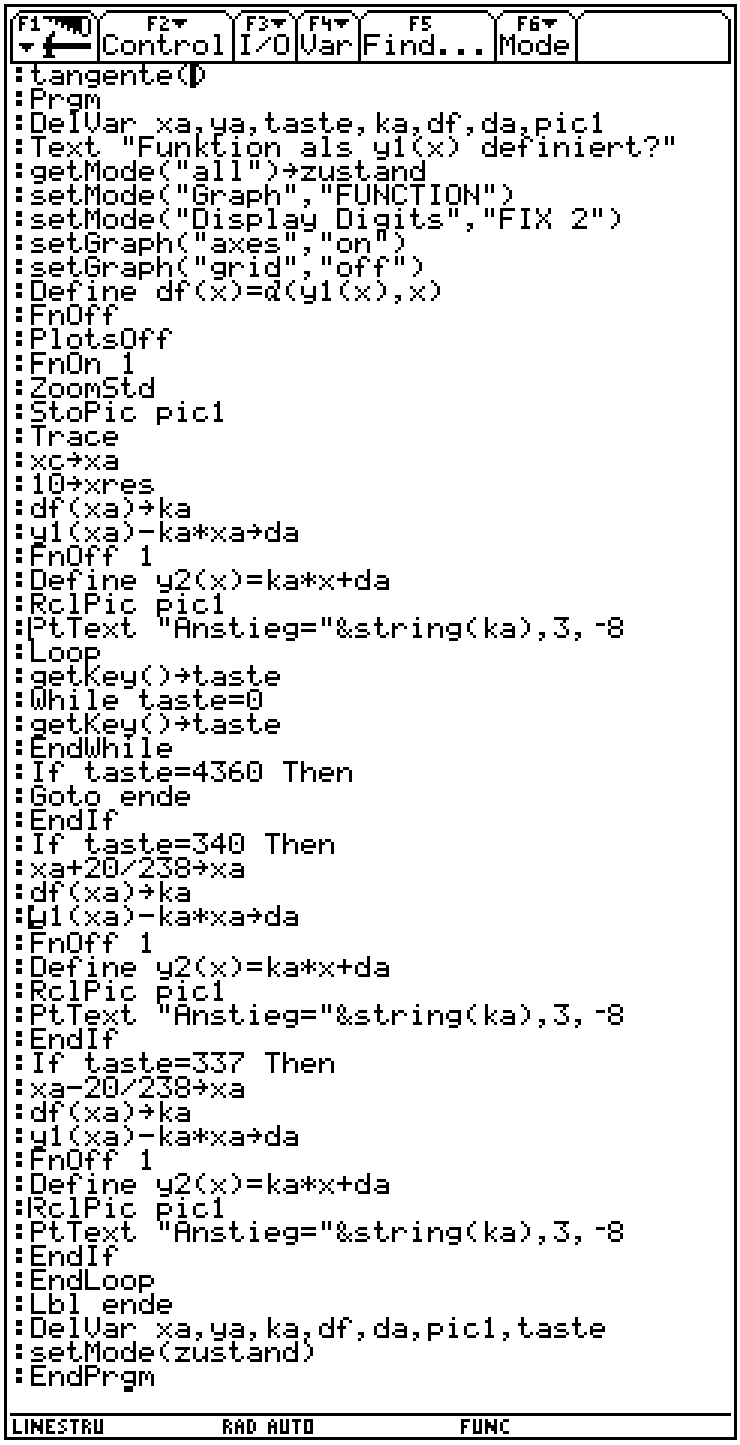
Mit ◁ oder ▷ können wir dem Graphen entlang fahren, wobei in jeder Position die Tangente gezeichnet und deren Anstieg ausgegeben wird.



Mit 2nd ESC kann das Programm beendet werden.

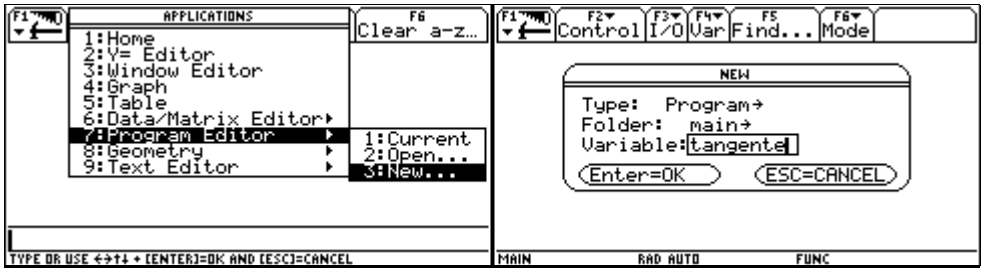


Programmcode:

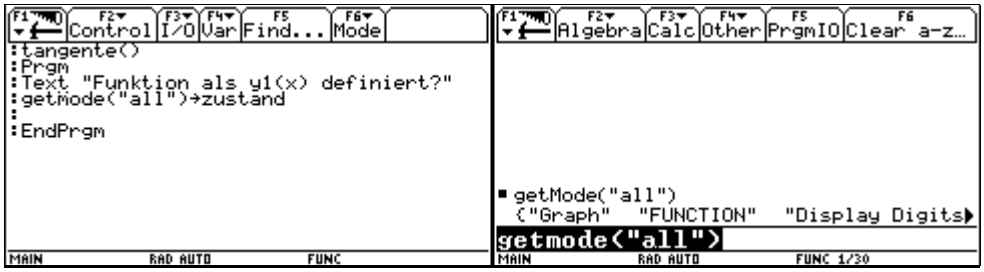


Erstellen des Programms:

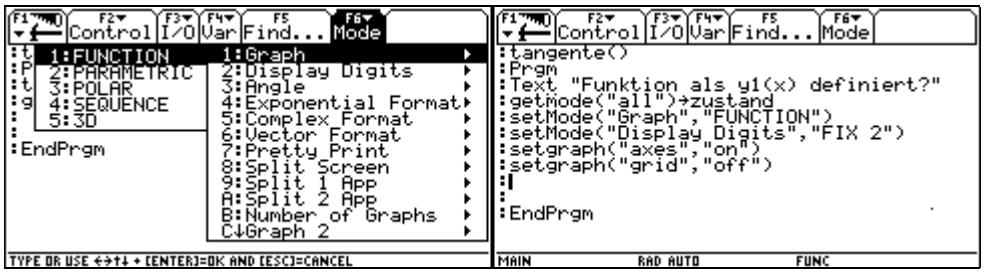
Wir beginnen im Programmeditor ein neues Programm mit dem Namen tangente.



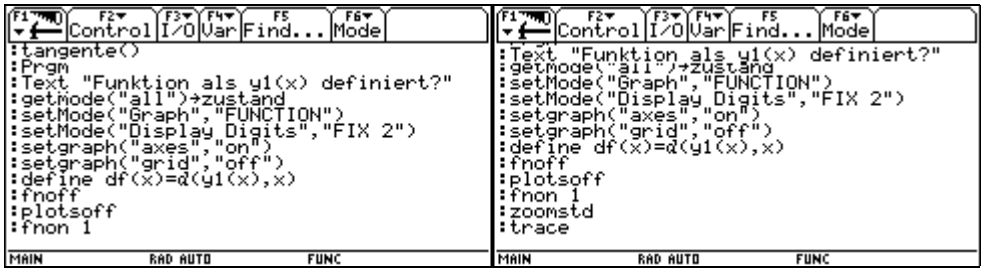
Zunächst geben wir die Frage ein, ob die Funktion auch als y1(x) gespeichert wurde. Dann speichern wir den Istzustand der MODE-Einstellungen unter der Variablen zustand ab. Dabei wird, wie man im Homebereich leicht überprüfen kann, eine lange Liste aller Einstellungen angelegt.



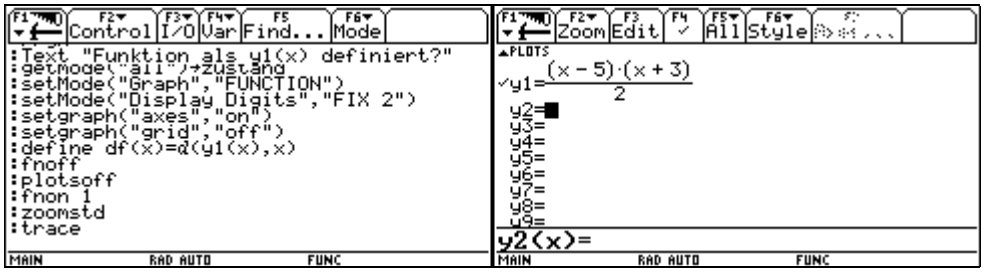
Dann legen wir die für dieses Programm notwendigen MODE-Einstellungen und Graphikeinstellungen fest.



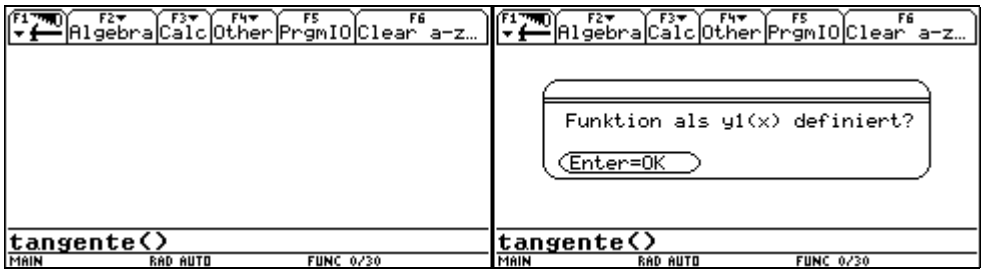
Es folgt das Deaktivieren aller Funktions- und Plotdefinitionen im [Y=]-Editor und das Aktivieren der Funktion y1(x). Außerdem wird die 1. Ableitung von y1(x) als df(x) definiert, die [WINDOW]-Variablen werden auf ZoomStd gesetzt und der Befehl trace aufgerufen.



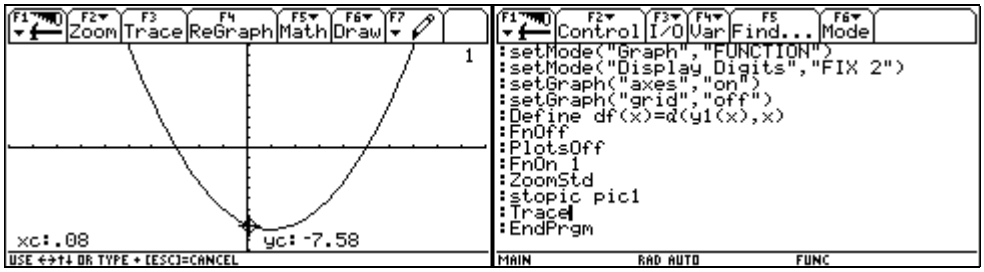
Wir steigen aus dem Programm aus und definieren eine Funktion y1(x) im [Y=]-Editor.



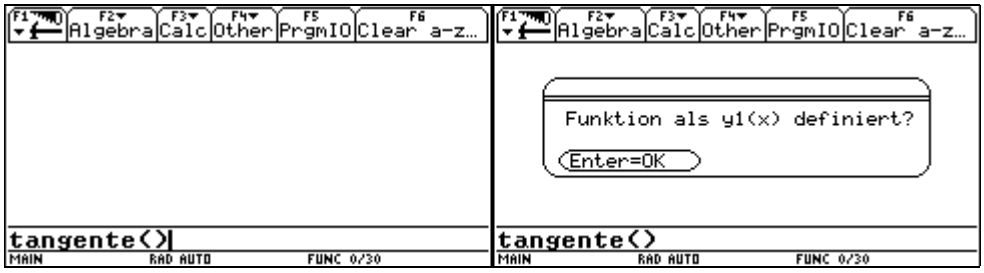
Danach starten wir das Programm und erhalten zunächst den schon bekannten Hinweis.



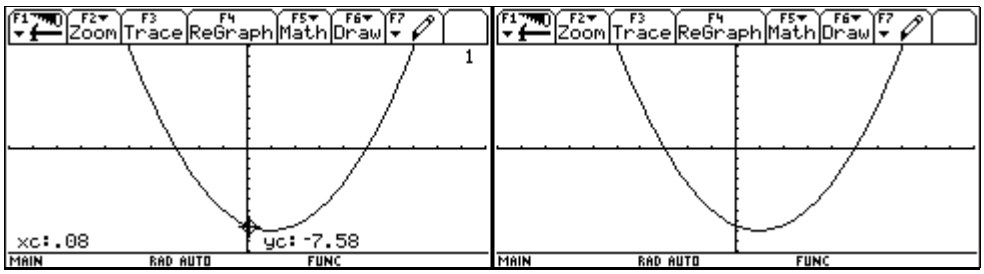
Darauf wird der Graph gezeichnet und der Graphikcursor sitzt, wie bei Trace üblich am Graphen. Da der Graph nicht immer neu gezeichnet werden soll, wenn die Tangente wandert, verändern wir unser Programm dahingehend, dass wir mit **StoPic** das Bild des Funktionsgraphen unter dem Variablennamen pic1 speichern.



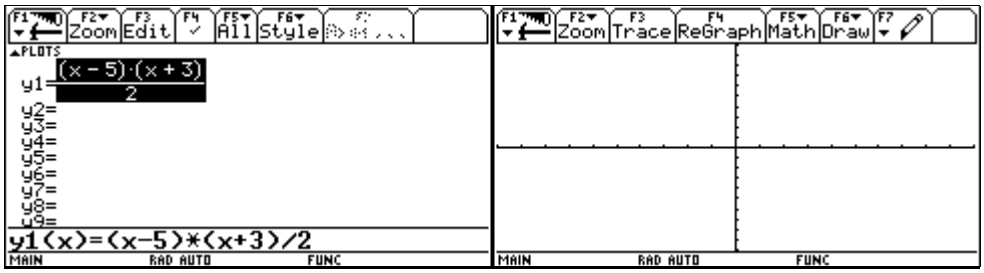
Nochmals starten wir das Programm.



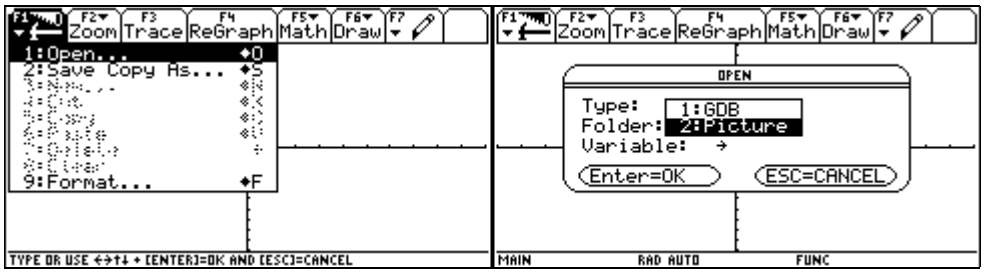
Wir bestätigen die Position des Cursors mit [ENTER] und beenden damit das Programm.

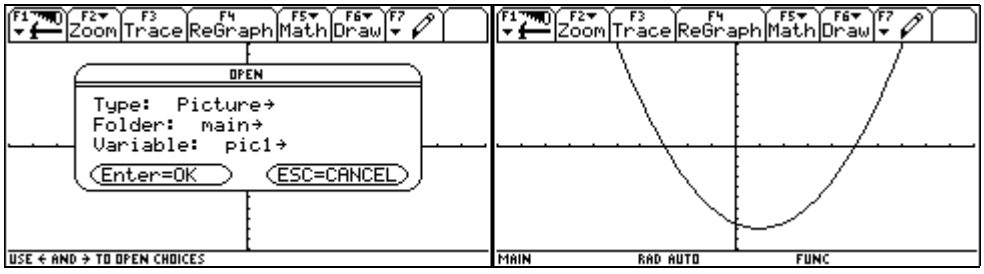


Im [Y=]Editor deaktivieren wir y1(x) und können über [GRAPH] den leeren Graphikschirm aufrufen.

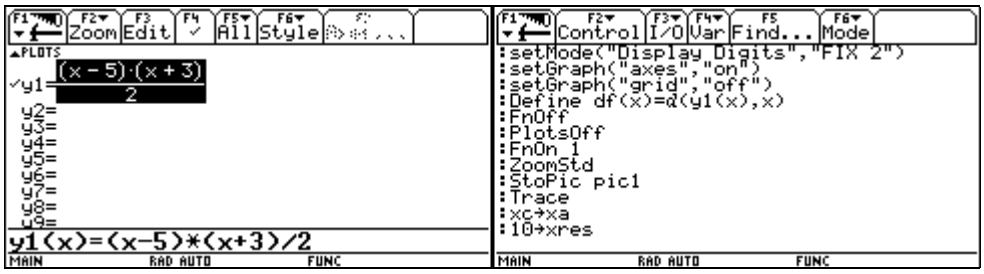


Über den Menüpunkt [F1] können wir nun unser gespeichertes Bild auf den Graphikschirm bringen.

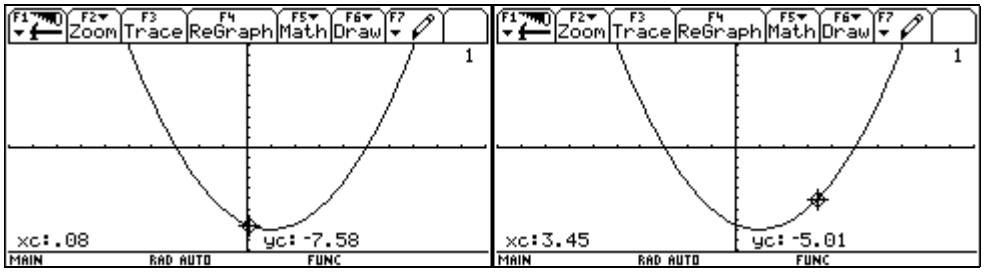




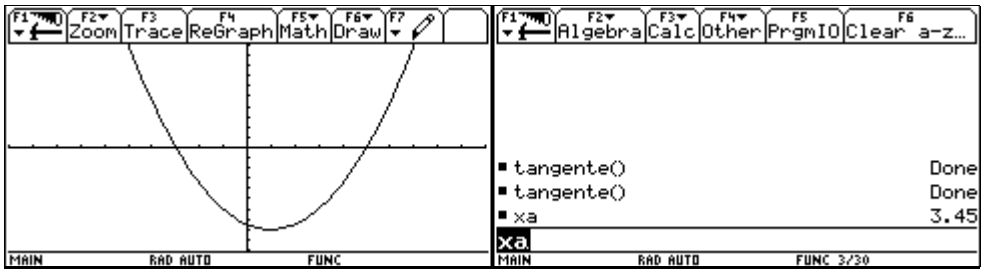
Wir aktivieren y1(x) wieder und kehren in das Programm zurück. Wenn wir im Trace-Zustand [ENTER] drücken, wird der Wert der *x*-Koordinate des Graphikcursors in **der Systemvariablen** xc abgespeichert. Diesen Wert halten wir in der Variablen xa fest. Die **Systemvariable** xres stellen wir auf 10, da ab nun nur noch Gerade gezeichnet werden, und dadurch die Zeichengeschwindigkeit drastisch erhöht wird.



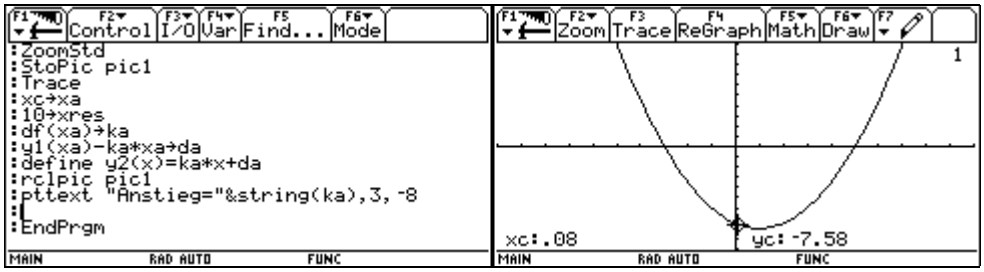
Wieder starten wir das Programm, um diese Neuerungen zu testen. Wir legen mit ENTER im Trace-Zustand eine *x*-Position fest.

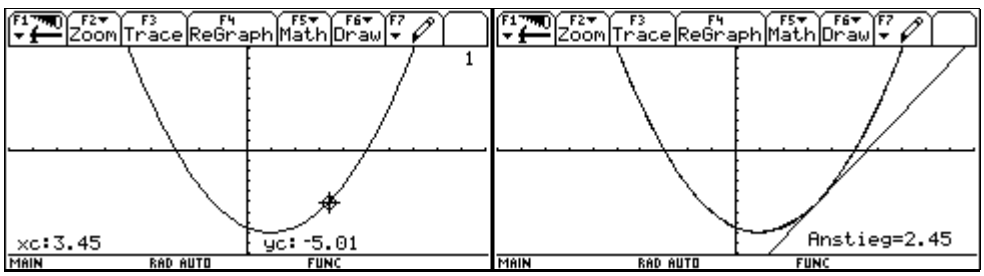


Nach Beendigung des Programms besitzt die Variable *xa* den festgelegten Wert.

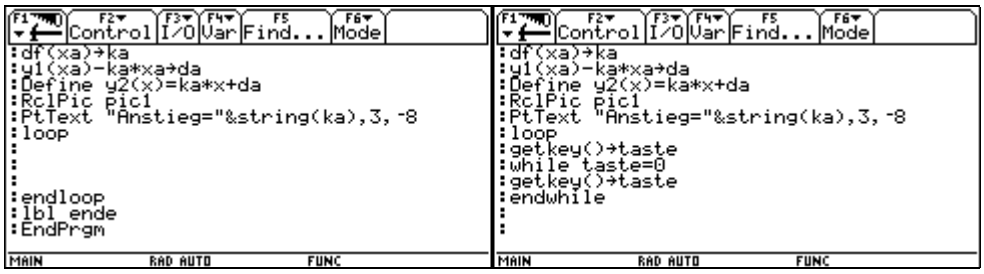


Zurück im Programm berechnen wir den Anstieg der Tangente *ka* an der Stelle *xa* und mit *da* den Abschnitt der Tangente auf der *y*-Achse. Die Gleichung der Tangente definieren wir als Funktion y2(x). Dann holen wir mit **RclPic** das Bild des gespeicherten Graphen zurück und schreiben mit dem Befehl PtText den Wert des Anstieges an einer geeigneten Stelle in die Zeichnung. Wir probieren wieder aus: wir fixieren einen Punkte am Graphen und erhalten die Tangente samt Anstieg.

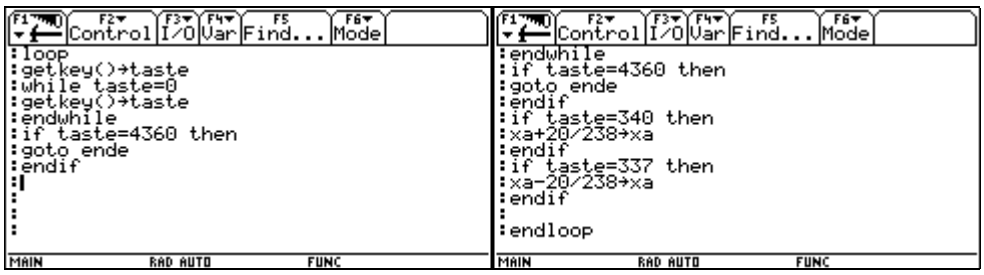




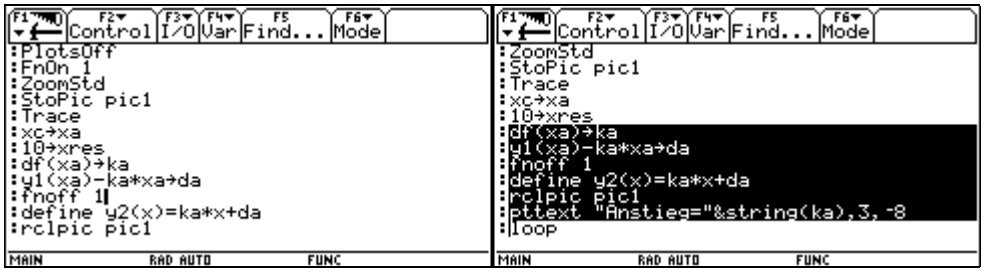
Nun wollen wir die Tangente den Graphen entlang bewegen. Wir schaffen eine große Schleife mit Loop-EndLoop. Da diese immer wieder durchlaufen wird, muss man in ihr eine Abbruchbedingung einbauen, wenn man keine Endlosschleife erzeugen möchte: Jeder Taste des Rechners ist eine Zahl zugeordnet. Die Funktion **getKey()** ermittelt den Code der gedrückten Taste. In einer **While-EndWhile** Schleife wird die Funktion getkey() solange aufgerufen, bis sie der Variablen taste einen Tastaturcode ≠ 0 übergeben hat. Wird nun eine Taste gedrückt, dann wird die While-**Schleife** verlassen und man kann über die in der Variablen taste abgespeicherte Zahl die gedrückte Taste ermitteln.



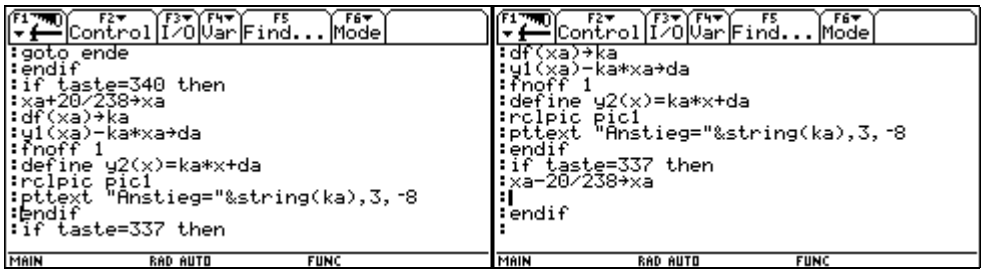
Zunächst wollen wir nicht auf den Abbruch unserer Loop-Schleife vergessen. Die Zahl 4360 ist der Tastenkombination 2nd ESC zugeordnet. In diesem und nur in diesem Fall soll die Loop-Schleife verlassen und zum Lbl ende gesprungen werden.
Der Taste ◁ entspricht der Code 340, der ▷ der Code 337. Je nachdem welche Taste gedrückt wurde soll der Wert der Variablen *xa* erhöht oder erniedrigt werden. Die hier verwendete Veränderung entspricht genau einem Pixelschritt bei der Einstellung ZoomStd.



Nun kopieren wir mit ♦ C die Befehl zur Berechnung und Zeichnung der Tangente in den Zwischenspeicher und ....

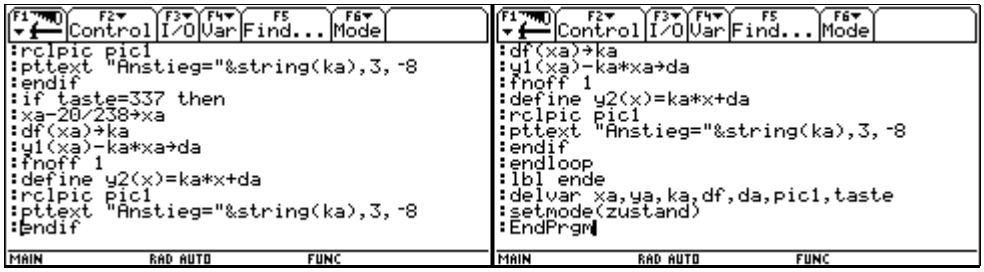


.... fügen diesen Befehlsblock an der entsprechenden Stelle mit ♦ V ein (nach der Veränderung der Variablen *xa* durch das Cursorpad).



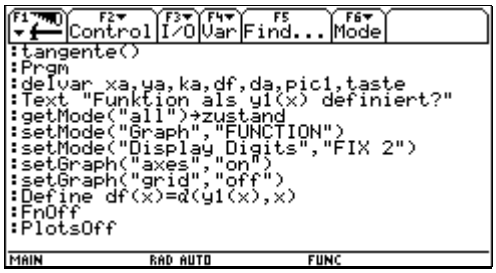
```
:goto ende
:endif
:if taste=340 then
:xa+20/238→xa
:df(xa)→ka
:y1(xa)-ka*xa→da
:fnoff 1
:define y2(x)=ka*x+da
:rclpic pic1
:pttext "Anstieg="&string(ka),3,-8
:endif
:if taste=337 then
:df(xa)→ka
:y1(xa)-ka*xa→da
:fnoff 1
:define y2(x)=ka*x+da
:rclpic pic1
:pttext "Anstieg="&string(ka),3,-8
:endif
:if taste=337 then
:xa-20/238→xa
:endif
```

Nach dem Lbl ende löschen wir alle Variablen und stellen den alten Systemzustand (MODE) wieder her.



```
:rclpic pic1
:pttext "Anstieg="&string(ka),3,-8
:endif
:if taste=337 then
:xa-20/238→xa
:df(xa)→ka
:y1(xa)-ka*xa→da
:fnoff 1
:define y2(x)=ka*x+da
:rclpic pic1
:pttext "Anstieg="&string(ka),3,-8
:endif
:df(xa)→ka
:y1(xa)-ka*xa→da
:fnoff 1
:define y2(x)=ka*x+da
:rclpic pic1
:pttext "Anstieg="&string(ka),3,-8
:endif
:endloop
:lbl ende
:delvar xa,ya,ka,df,da,pic1,taste
:setmode(zustand)
:EndPrgm
```

Auch zu Beginn dieses Programms löschen wir die vorkommenden Variablen.



```
:tangente()
:Prgm
:delvar xa,ya,ka,df,da,pic1,taste
:Text "Funktion als y1(x) definiert?"
:getMode("all")→zustand
:setMode("Graph","FUNCTION")
:setMode("Display Digits","FIX 2")
:setGraph("axes","on")
:setGraph("grid","off")
:Define df(x)=d(y1(x),x)
:FnOff
:PlotsOff
```

Lehrinhalte: Programmierung mit Hilfe der Graphikcursorkoordinaten, Steuerung über die Tastatur, die Systemvariablen xc und yc, Dauerschleife, Laden und Speichern von Bildern

Befehle: DelVar, Text, getMode, setMode, setGraph, Define, FnOff,FnOn, PlotsOff, ZoomStd, StoPic, RclPic, Trace, PtText, Loop-EndLoop, While-EndWhile, getkey()

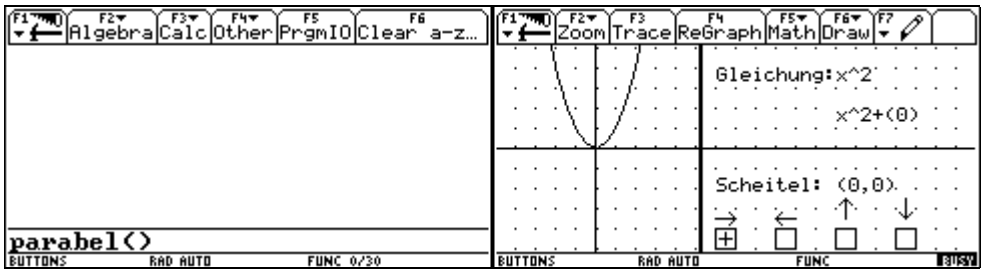
Übung: Darstellung der Fläche unter einem Graphen bei schrittweisem Entlangfahren am Graphen

2.6 Steuerung über Schaltflächen

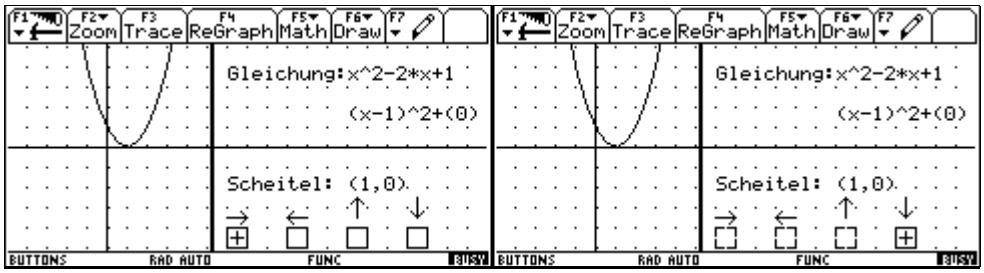
Programm: Verschiebung des Scheitels einer Parabel und Anzeige der entsprechenden Parabelgleichung durch Steuerung über zwei Schaltflächen

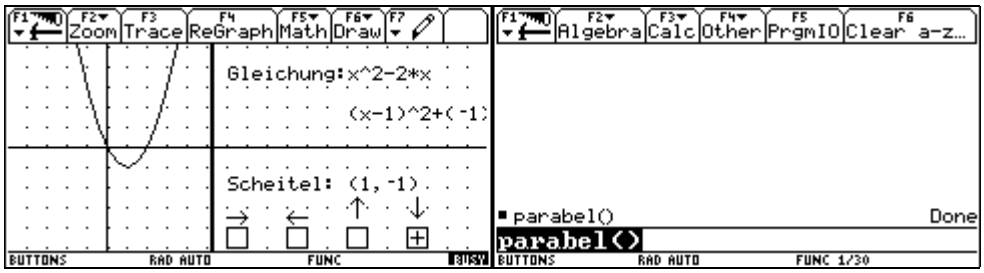
Benutzung:

Nach dem Start des Programms erhalten wir das folgende Bild:



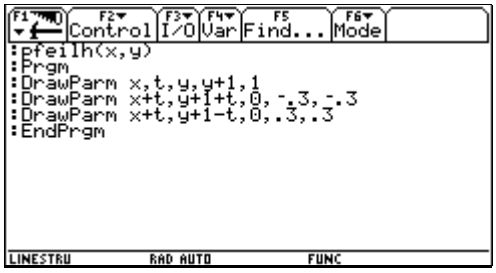
Mit Hilfe des Cursorpads ◁ bzw. ▷ kann der kreuzförmige Cursor zwischen den Schaltflächen bewegt werden. Mit der ENTER-Taste wird der Parabelscheitel in der angezeigten Pfeilrichtung um eine Einheit verschoben. Gleichzeitig werden die neue Gleichung und die Koordinaten des neue Scheitels angezeigt. Mit 2nd ESC wird das Programm verlassen.

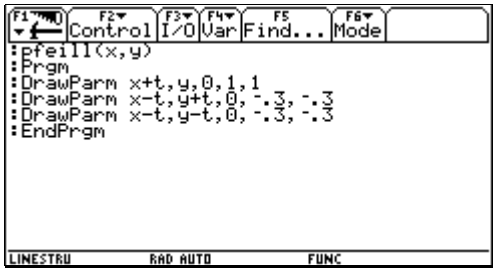


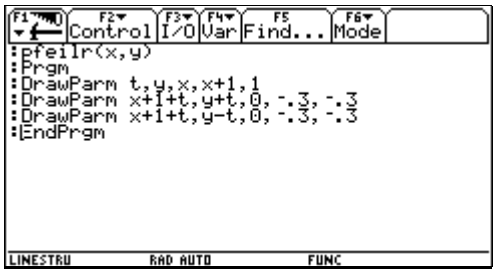


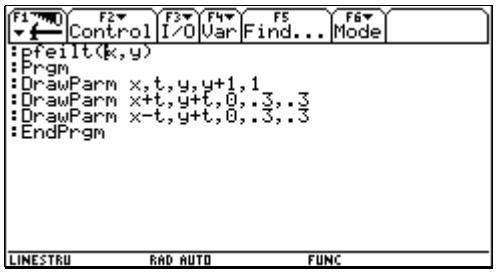
Programmcode:

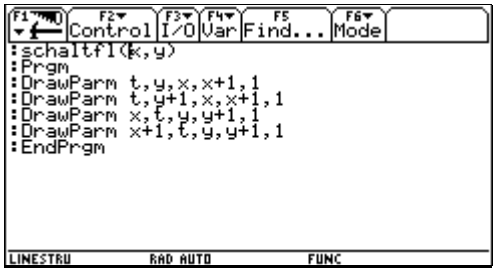
7 Hilfsprogramme (Unterprogramme):

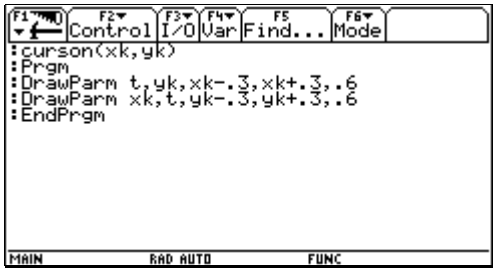


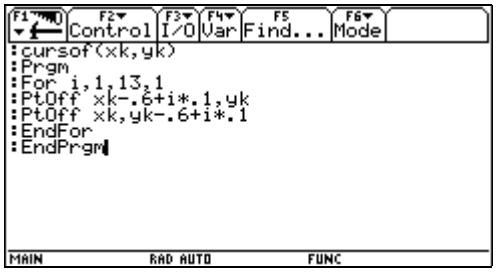












Die Funktionsweise der Hilfsprogramme wird im Anschluss näher erklärt. Sie werden vom Hauptprogramm aus gesteuert und aufgerufen.

Hauptprogramm:

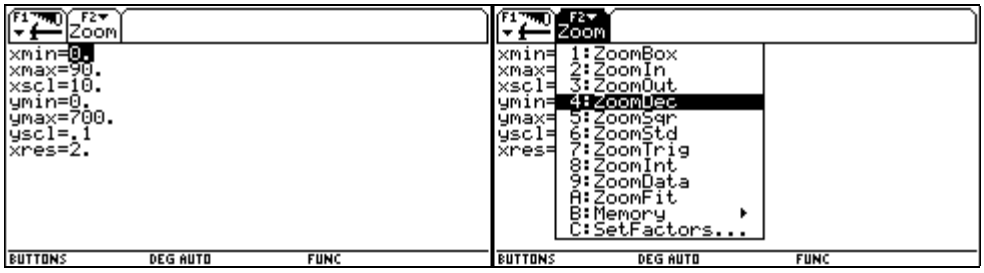
```
F1 F2 Control F3 I/O F4 Var F5 Find... F6 Mode
:parabel()
:Prgm
:DelVar px,py,xk,yk,gl,i,pic1,taste,a
:0→px
:0→py
:-4.9→xmin
:18.9→xmax
:-5.1→ymin
:5.1→ymax
:1→xscl
:1→yscl
:5→xres
:getMode("all")→zustand
:setMode("Graph","FUNCTION")
:setMode("Split Screen","FULL")
:setGraph("axes","on")
:setGraph("grid","on")
:FnOff
:PlotsOff
:schaltfl(6,-5)
:schaltfl(9,-5)
:schaltfl(12,-5)
:schaltfl(15,-5)
:pfeilr(6,-3.5)
:pfeill(9,-3.5)
:pfeilh(12.5,-3.5)
:pfeilt(15.5,-3.5)
:PtText "Scheitel:",6,-1.5
:PtText "Gleichung:",6,4
:DrawParm 5.2,t,-6,6,12
:DrawParm 5.3,t,-6,6,12
:StoPic pic1
:setGraph("axes","off")
:setGraph("grid","off")
:6.5→xk
:-4.5→yk
:Lbl anfang
:Define y1(x)=(x-px)^2+py
:FnOn 1
:RclPic pic1
:curson(xk,yk)
:(x-px)^2→a
:expand((x-px)^2+py)→gl
:PtText string(gl),12,4
:PtText string(a)&"+("&string(py)&")",12
 ,2
:PtText "("&string(px)&","&string(py)&")
 ",12,-1.5
:Loop
:getKey()→taste
:While taste=0
:getKey()→taste
:EndWhile
:If taste=4360 Then
:Goto ende
:EndIf
:If taste=340 and xk<15.5 Then
:cursof(xk,yk)
:curson(xk+3,yk)
:xk+3→xk
:EndIf
```

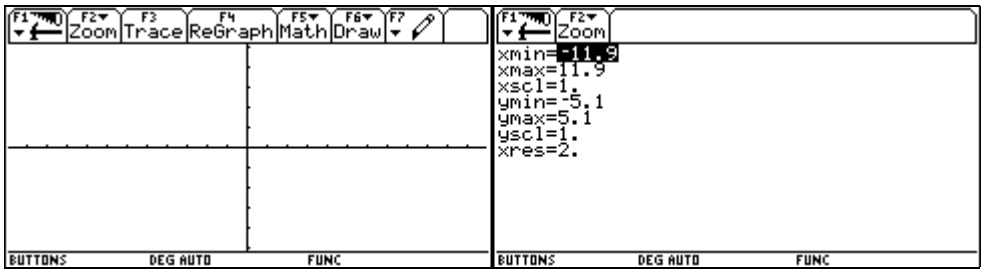
```
:If taste=337 and xk>6.5 Then
:cursof(xk,yk)
:curson(xk-3,yk)
:xk-3→xk
:EndIf
:If taste=13 Then
:If xk=6.5 Then
:px+1→px
:EndIf
:If xk=9.5 Then
:px-1→px
:EndIf
:If xk=12.5 Then
:py+1→py
:EndIf
:If xk=15.5 Then
:py-1→py
:EndIf
:Goto anfang
:EndIf
:EndLoop
:Lbl ende
:setMode(zustand)
:DelVar px,py,xk,yk,gl,i,pic1,taste,a
:EndPrgm
```

LINESTRU RAD AUTO FUNC

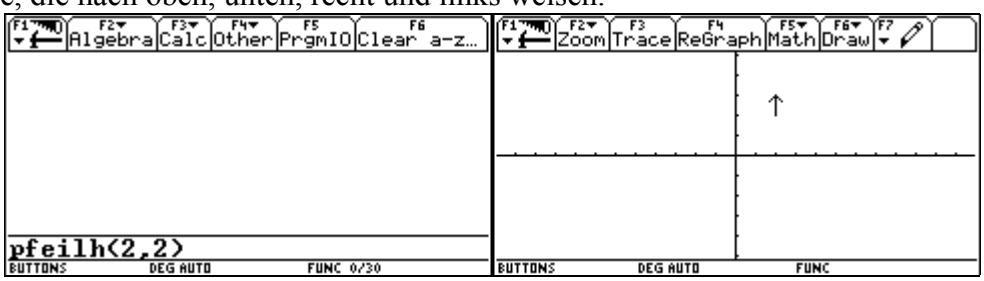
Erstellung des Programms:

Zuerst wollen wir die Wirkung der Hilfsprogramme testen. Dazu stellen wir den Graphikschirm auf die Einstellung ZoomDec.

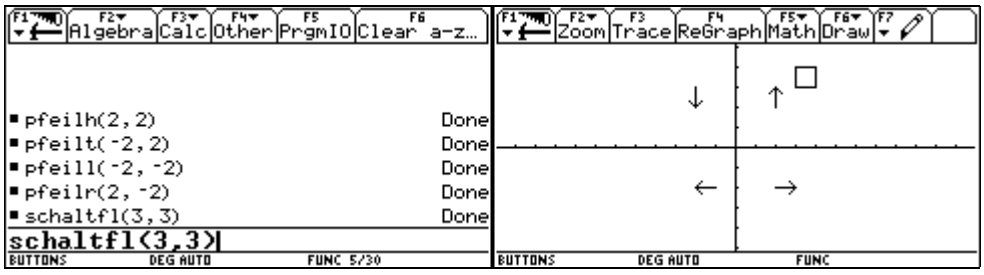




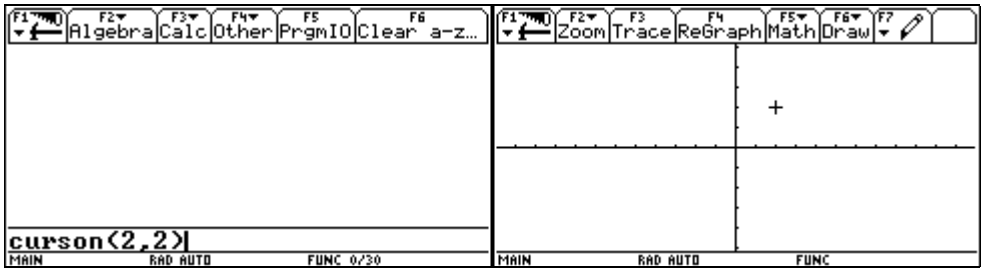
Die Hilfsprogramme mit den Namen pfeilh, pfeilt, pfeilr und pfeill zeichnen Pfeile, die nach oben, unten, recht und links weisen.



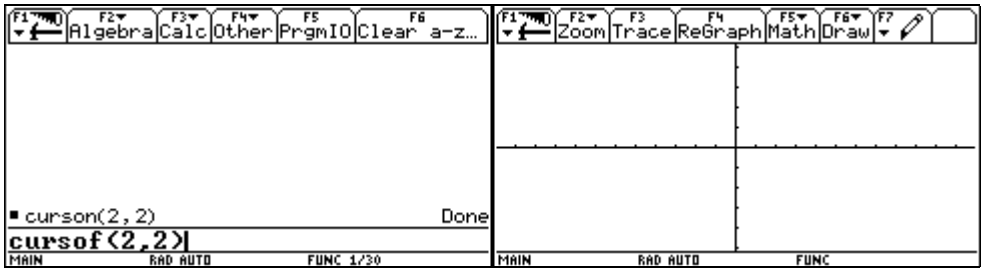
Das Hilfsprogramm schaltfl zeichnet ein Quadrat an die entsprechende Stelle.



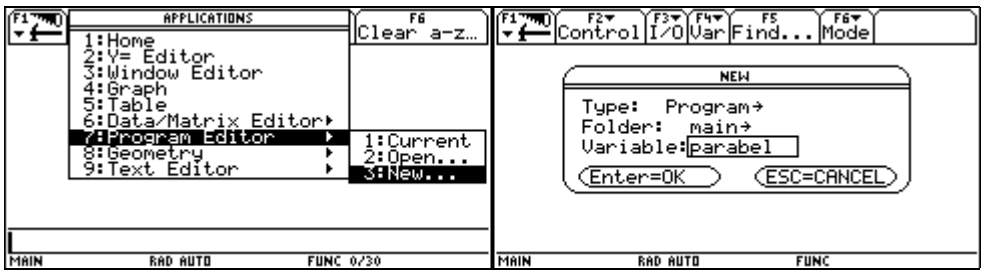
curson(x,y) erzeugt einen kreuzförmigen Cursor an der durch die Koordinaten *x* und *y* angegebenen Position.



cursof(x,y) löscht das Bild des kreuzförmigen Cursors an der Position (*x*,*y*).

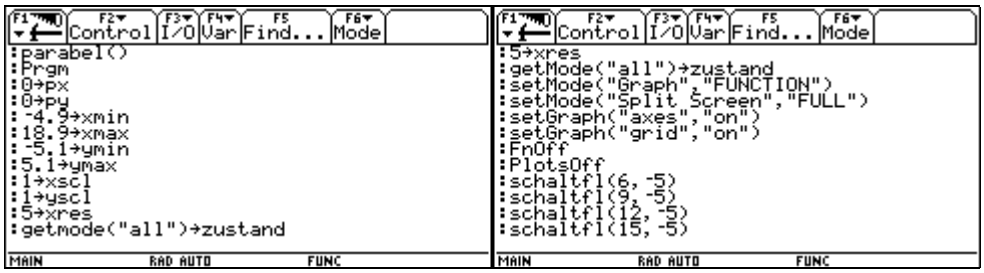


Wir öffnen den Programmeditor und geben dem neuen Programm den Namen parabel.

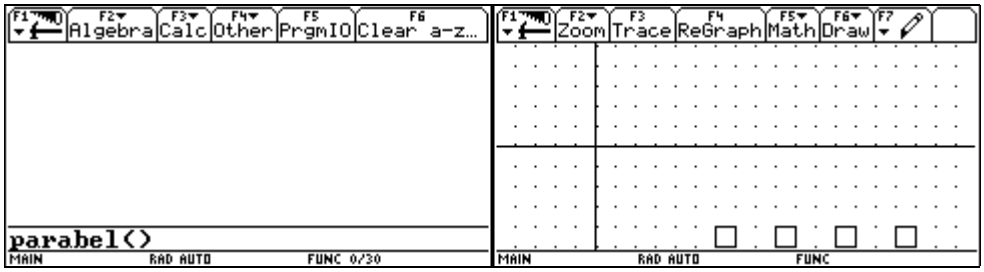


Die Scheitelkoordinaten *px* und *py* werden beide Null gesetzt. Das Graphikfenster erhält die Ausmaße gemäß der ZoomDec-Einstellung aber in Richtung der *x*-Achse verschoben. Die Skalierungen werden auf 1 gesetzt. **xres** erhält den Wert 5, was für die die Darstellung von Parabeln ausreichend ist. Der Istzustand der MODE-Einstellungen wird gespeichert. Danach werden die gewünschten Grundeinstellungen

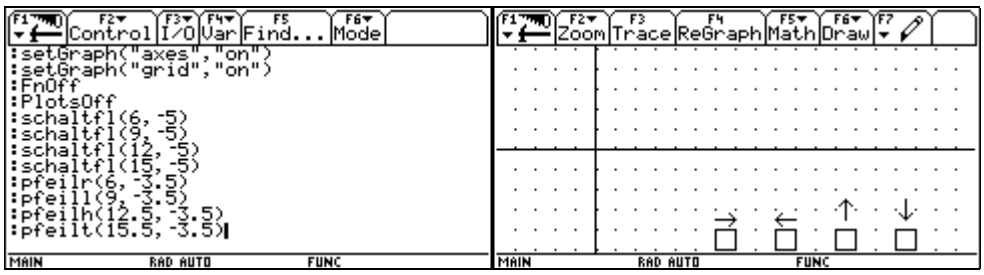
für MODE und Graphik vorgenommen. Alle Funktionen und Plots werden deaktiviert. Anschließend werden die Schaltflächen angelegt.



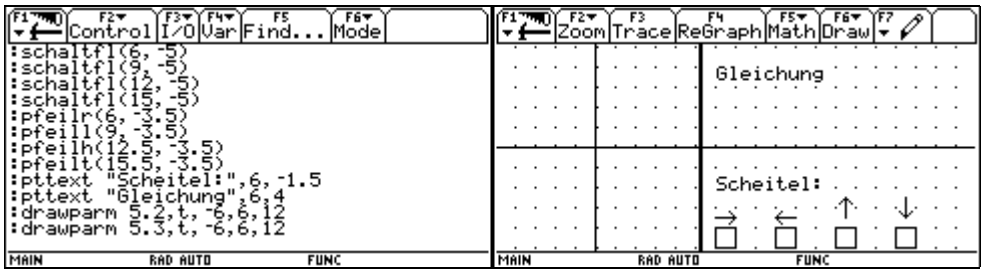
Durch Starten des Programms könne wir die Lage der Schaltflächen überprüfen.



Ebenso wird bei der Positionierung der Pfeile vorgegangen.

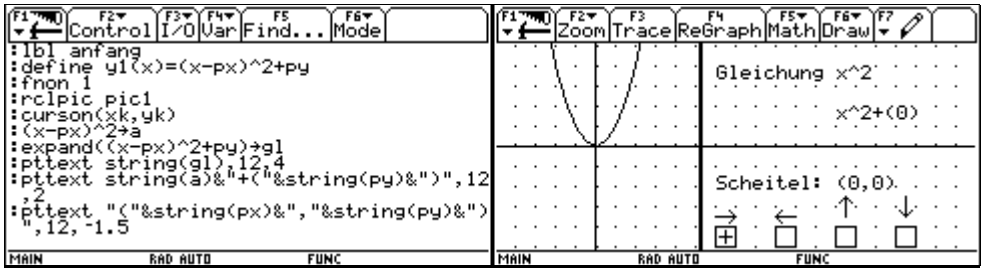


Danach wird die Lage der Texte "Gleichung" und "Scheitel" programmiert. Außerdem wird eine Doppellinie angelegt, die den Graphen optisch von den Schaltflächen trennt.

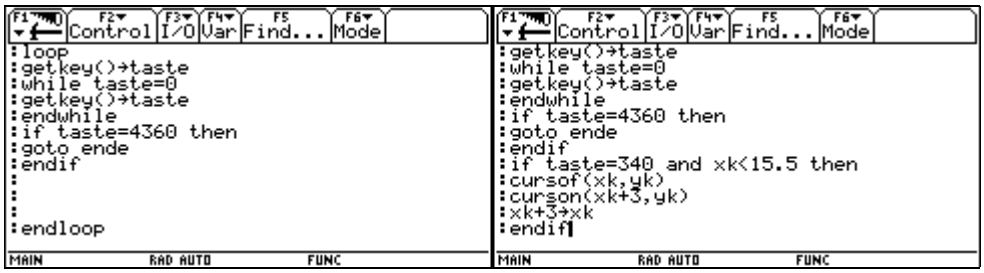


Dieses Bild wird unter pic1 gespeichert und das Zeichnen von Koordinatenachsen und Gitterpunkten wird ausgeschaltet. Ferner werden die Koordinaten des kreuzförmigen Cursors auf ihre Anfangswerte gesetzt. Dann wird das Lbl anfang gesetzt, zu dem immer wieder verzweigt wird, wenn die Zeichnung erneuert werden soll. Die Gleichung der Parabel wird in y1(x) abgelegt und y1(x) wird aktiviert. Das gespeicherte Hintergrundbild wird abgerufen und der kreuzförmige Cursor an die aktuelle Position gesetzt.
Um auch die quadratische Ergänzung zur Scheitelbestimmung zeigen zu könne, wird der quadratische Anteil der Gleichung unter *a* abgespeichert. Der Term der Gleichung

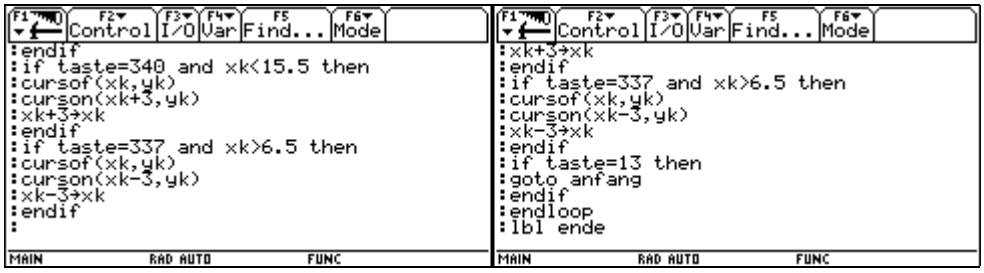
wird berechnet, unter *gl* gespeichert und an geeigneter Stelle angezeigt. Dazu werden die Ergänzung auf ein vollständiges Quadrat und die Scheitelkoordinaten ausgegeben.



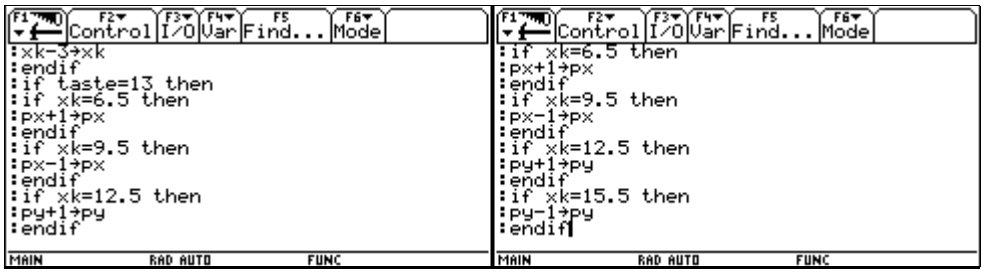
Zur Steuerung über die Tastatur werden wieder eine Loop- und eine While-Schleife angelegt, wie wir dies schon im Kapitel über die Steuerung durch die Tastatur kennengelernt haben. Über die Tastenfolge 2nd ESC (Code 4360) wird das Ende des Programms angesteuert. Mit ⊙ (Code 340) wird der kreuzförmige Cursor zur nächsten Schaltfläche rechts bewegt, außer wenn er sich schon ganz rechts befindet. Analog bewegt man den kreuzförmige Cursor mit ⊙ (Code 337) zur nächsten Schaltfläche links, außer wenn er sich schon ganz links befindet.



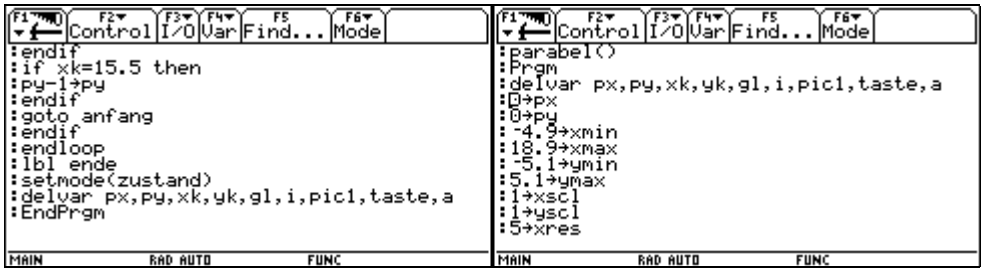
Wenn die Taste ENTER (Code 13) gedrückt wird, dann soll neu gezeichnet werden, also wird zum Lbl anfang gesprungen.



Dabei sind die Scheitelkoordinaten je nach Position des kreuzförmigen Cursors zu verändern.



Abschließend werden die Variablen am Beginn und zu Ende des Programms gelöscht. Die MODE-Einstellungen werden wieder zurückgestellt.



```
:endif
:if xk=15.5 then
:py-1→py
:endif
:goto anfang
:endif
:endloop
:lbl ende
:setmode(zustand)
:delvar px,py,xk,yk,gl,i,pic1,taste,a
:EndPrgm
```

```
:parabel()
:Prgm
:delvar px,py,xk,yk,gl,i,pic1,taste,a
:0→px
:0→py
:-4.9→xmin
:18.9→xmax
:-5.1→ymin
:5.1→ymax
:1→xscl
:1→yscl
:5→xres
```

Lehrinhalte: Erstellen von graphischen Schaltflächen (Buttons), Bewegung des Graphikcursors und Abfragen seiner Position, Anzeigen von Text am Graphikschirm

Befehle: Loop-EndLoop, DrawParm, PtText, While-EndWhile, setMode, setGraph, getMode, Define, expand, DelVar, Goto, If-Then-EndIf, Lbl, getKey(), PlotsOff, FnOff, FnOn, StoPic, RclPic

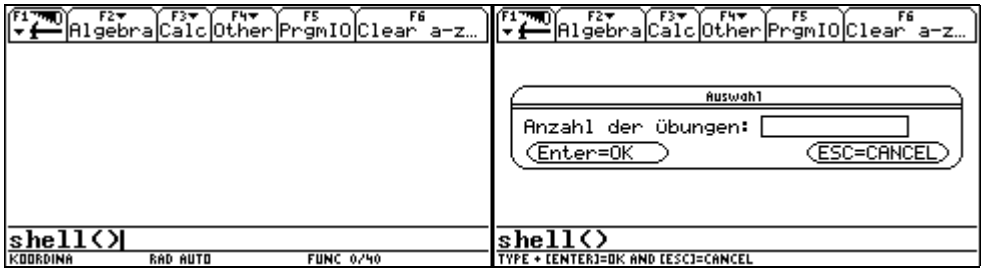
Übung: Veränderung von Kreisfrequenz und Phasenverschiebung einer verallgemeinerten Sinsufunktion und Anzeige der entsprechenden Funktionsgleichung durch Steuerung über zwei Buttons.

2.7 Steuerung über Zufallsvariable

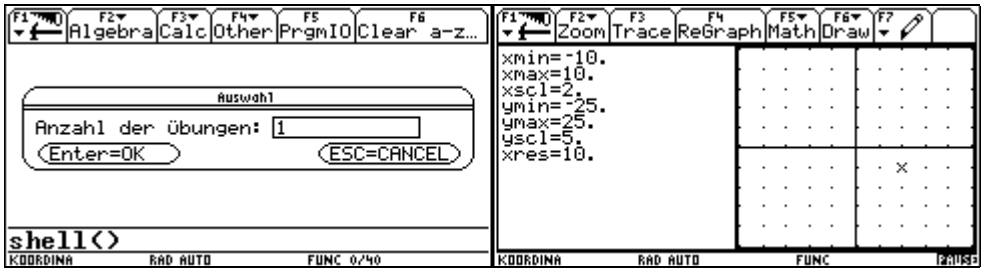
Programm: Übungsprogramm zum Ablesen von Punktkoordinaten

Benutzung:

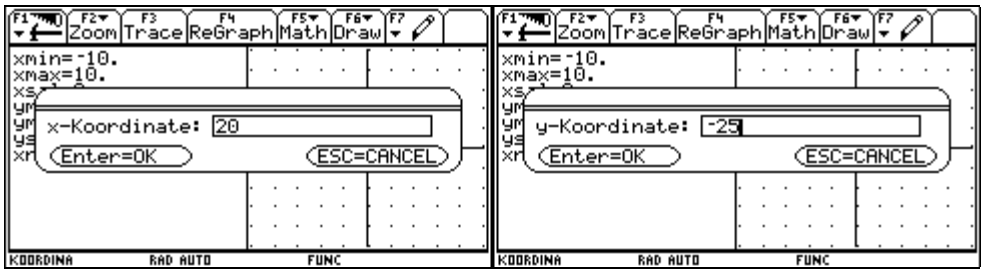
Nach dem Start ist die Anzahl der Übungen festzulegen.



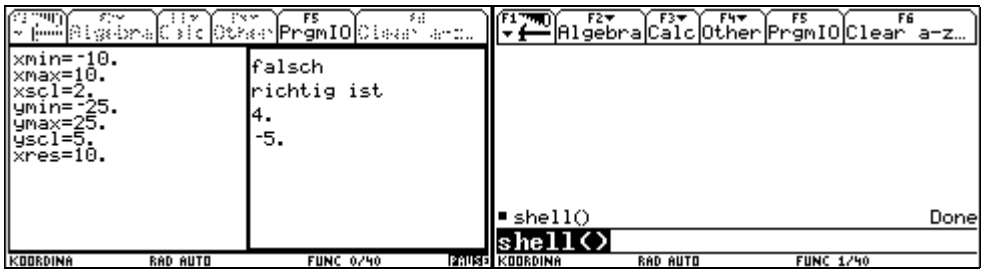
Danach teilt sich der Bildschirm. Links sind die [WINDOW]-Variablen und rechts ein Koordinatensystem mit einem markierten Gitterpunkt zu sehen. Das Programm steht auf PAUSE, um Zeit zum Ablesen der Koordinaten zu geben.



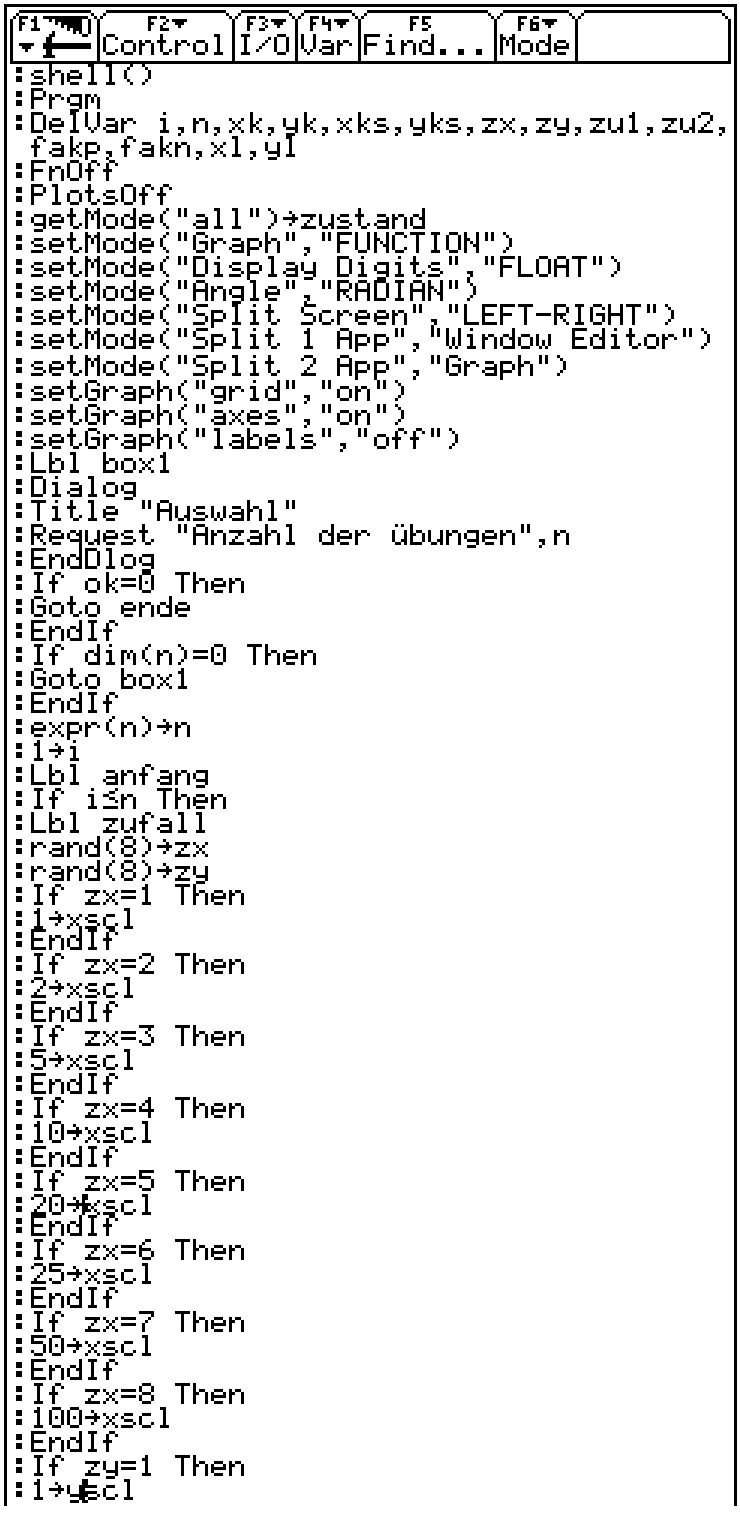
Danach wird man aufgefordert *x*- und *y*-Koordinate des markierten Punktes einzugeben.



Es erfolgt die Kontrolle und gegebenenfalls die Verbesserung des Resultates. Nach der Beendigung der PAUSE wird das Programm beendet.



Programmcode:



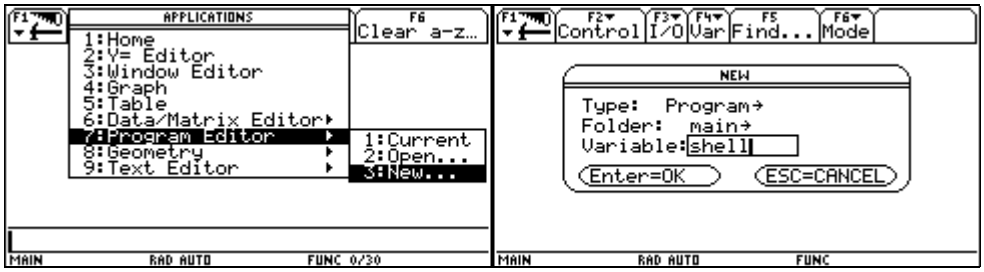
```
F1 F2 Control F3 I/O F4 Var F5 Find... F6 Mode
:shell()
:Prgm
:DelVar i,n,xk,yk,xks,yks,zx,zy,zu1,zu2,
 fakp,fakn,xl,yl
:FnOff
:PlotsOff
:getMode("all")→zustand
:setMode("Graph","FUNCTION")
:setMode("Display Digits","FLOAT")
:setMode("Angle","RADIAN")
:setMode("Split Screen","LEFT-RIGHT")
:setMode("Split 1 App","Window Editor")
:setMode("Split 2 App","Graph")
:setGraph("grid","on")
:setGraph("axes","on")
:setGraph("labels","off")
:Lbl box1
:Dialog
:Title "Auswahl"
:Request "Anzahl der Übungen",n
:EndDlog
:If ok=0 Then
:Goto ende
:EndIf
:If dim(n)=0 Then
:Goto box1
:EndIf
:expr(n)→n
:1→i
:Lbl anfang
:If i≤n Then
:Lbl zufall
:rand(8)→zx
:rand(8)→zy
:If zx=1 Then
:1→xscl
:EndIf
:If zx=2 Then
:2→xscl
:EndIf
:If zx=3 Then
:5→xscl
:EndIf
:If zx=4 Then
:10→xscl
:EndIf
:If zx=5 Then
:20→xscl
:EndIf
:If zx=6 Then
:25→xscl
:EndIf
:If zx=7 Then
:50→xscl
:EndIf
:If zx=8 Then
:100→xscl
:EndIf
:If zy=1 Then
:1→yscl
```

```
:EndIf
:If zy=2 Then
:2→yscl
:EndIf
:If zy=3 Then
:5→yscl
:EndIf
:If zy=4 Then
:10→yscl
:EndIf
:If zy=5 Then
:20→yscl
:EndIf
:If zy=6 Then
:25→yscl
:EndIf
:If zy=7 Then
:50→yscl
:EndIf
:If zy=8 Then
:100→yscl
:EndIf
:rand(10)→fakp
:rand(-10)→fakn
:xscl*fakp→xmax
:xscl*fakn→xmin
:yscl*fakp→ymax
:yscl*fakn→ymin
:rand(2)→zu1
:If zu1=1 Then
:(rand(fakp)-1)*xscl→xk
:Else
:(rand(fakn)+1)*xscl→xk
:EndIf
:rand(2)→zu2
:If zu2=1 Then
:(rand(fakp)-1)*yscl→yk
:Else
:(rand(fakn)+1)*yscl→yk
:EndIf
:If xk=0 and yk=0 Then
:Goto zufall
:EndIf
:(xk)→xl
:(yk)→yl
:NewPlot 1,1,xl,yl,,,,2
:DispG
:Pause
:Lbl box2
:Request "x-Koordinate",xks
:If ok=0 Then
:Goto ende
:EndIf
:If dim(xks)=0 Then
:Goto box2
:EndIf
:Lbl box3
:Request "y-Koordinate",yks
:If ok=0 Then
:Goto ende
```

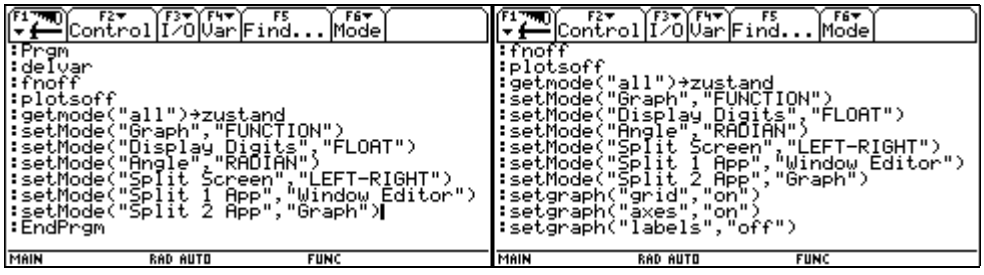
```
:EndIf
:If dim(yks)=0 Then
:Goto box3
:EndIf
:expr(xks)→xks
:expr(yks)→yks
:If xk=xks and yk=yks Then
:ClrIO
:Disp "richtig"
:Pause
:Else
:ClrIO
:Disp "falsch"
:Disp "richtig ist"
:Disp xk
:Disp yk
:Pause
:EndIf
:i+1→i
:Goto anfang
:EndIf
:Lbl ende
:setMode(zustand)
:PlotsOff
:DelVar i,n,xk,yk,xks,yks,zx,zy,zu1,zu2,
 fakp,fakn,xl,yl
:EndPrgm
MAIN          RAD AUTO          FUNC
```

Erstellung des Programms:

Wir öffnen im Programmeditor ein neues Programm mit dem Namen shell.



Gleich zu Beginn setzen wir den Befehl zum Löschen der Variablen. Dann deaktivieren wir allfällig vorhandene Funktionen und Plots, sichern den aktuellen Systemzustand und stellen [MODE] und Graphik für das Programm ein.



Danach erstellen wir die Dialogbox für die Eingabe der Anzahl der Übungen.

```
:setMode("Split 2 App","Graph")
:setgraph("grid","on")
:setgraph("axes","on")
:setgraph("labels","off")
:lbl box1
:Dialog
:title "Auswahl"
:request "Anzahl der Übungen",n
:EndDlog
:if ok=0 then
:goto ende
:endif
```

```
:title "Auswahl"
:request "Anzahl der Übungen",n
:EndDlog
:if ok=0 then
:goto ende
:endif
:if dim(n)=0 then
:goto box1
:endif
:
:lbl ende
:EndPrgm
```

Die Anzahl der Übungen wird mit expr aus dem eingegebenen String ermittelt. Die **Zählvariable *i*** für die Anzahl der bisher durchgeführten Übungen wird auf 1 gesetzt. Das Label anfang dient dazu, um nach Beendigung einer Übung zur nächsten weiterzugehen. Dann wird abgefragt, ob die vom Benutzer angegebene Anzahl von Übungen schon erreicht ist. Das Label zufall ermöglicht es, bei ungünstiger Wahl der Zufallszahlen, diese neu zu generieren. Mit dem Befehl **rand(8)** wird zufällig eine natürliche Zahl zwischen 1 und 8 erzeugt. Je nach Größe der Zufallszahlen wird nun die **Skalierung der Achsen** festgelegt.

```
:endif
:if dim(n)=0 then
:goto box1
:endif
:expr(n)→n
:1→i
:lbl anfang
:if i≤n then
:lbl zufall
:rand(8)→zx
:rand(8)→zy
:
```

```
:if zx=1 then
:1→xscl
:endif
:if zx=2 then
:2→xscl
:endif
:if zx=3 then
:5→xscl
:endif
:if zx=4 then
:10→xscl
:endif
```

```
:if zx=5 then
:20→xscl
:endif
:if zx=6 then
:25→xscl
:endif
:if zx=7 then
:50→xscl
:endif
:if zx=8 then
:100→xscl
:endif
```

```
:if zy=1 then
:1→yscl
:endif
:if zy=2 then
:2→yscl
:endif
:if zy=3 then
:5→yscl
:endif
:if zy=4 then
:10→yscl
:endif
```

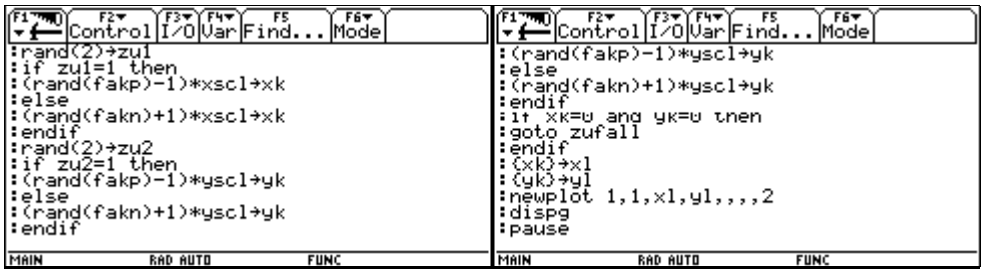
Zwei weitere **Zufallszahlen** fakp und fakn, eine positiv und eine negativ, legen mit Hilfe der schon bestimmten Skalierungen die Fenstereinstellungen für xmax, xmin, ymin, und ymax fest.

```
:if zy=5 then
:20→yscl
:endif
:if zy=6 then
:25→yscl
:endif
:if zy=7 then
:50→yscl
:endif
:if zy=8 then
:100→yscl
:endif
```

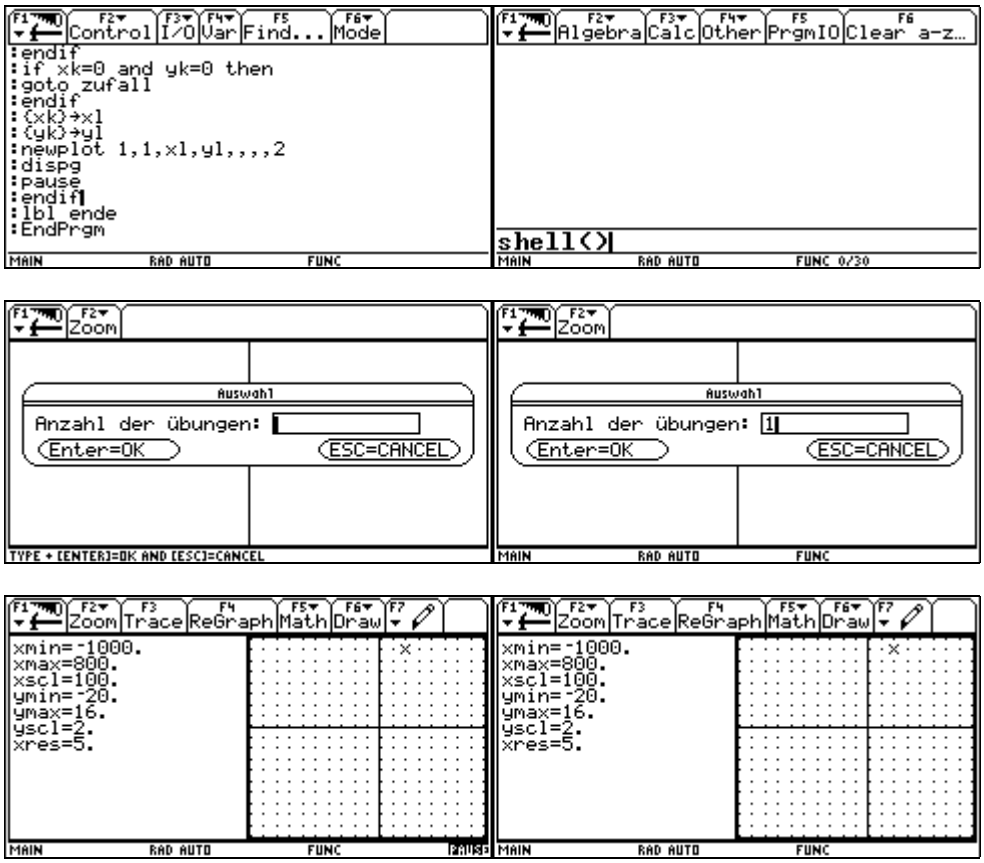
```
:if zy=7 then
:50→yscl
:endif
:if zy=8 then
:100→yscl
:endif
:rand(10)→fakp
:rand(-10)→fakn
:xscl*fakp→xmax
:xscl*fakn→xmin
:yscl*fakp→ymax
:yscl*fakn→ymin
```

Über die **Zufallszahl** *zu1* (1 oder 2) wird entschieden, ob die ***x*-Koordinate des angezeigten Gitterpunktes positiv oder negativ wird.** Dann wird eine Zufallszahl zwischen 1 und *fakp* bestimmt und 1 abgezogen, damit der Gitterpunkt nicht am Rand des Bildschirmes zu liegen kommt. Analog wird für die *y*-Koordinate vorgegangen. Dann wird über eine Abfrage verhindert, dass der Koordinatenursprung als Gitterpunkt gewählt wird. In diesem Fall wird zum Label zufall zurückgesprungen und die Zufallszahlen werden neu erzeugt. Zwei Listen werden angelegt, in denen die

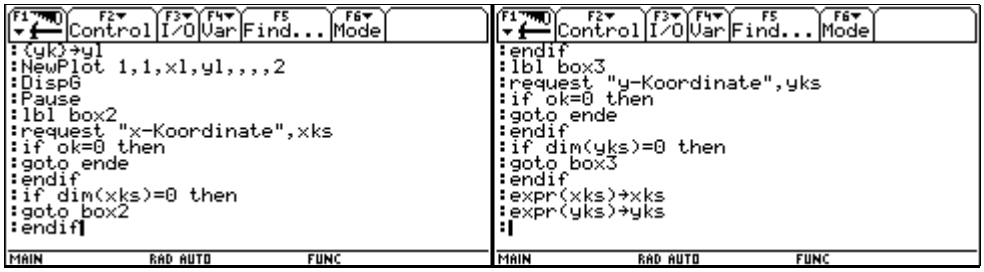
Koordinaten der Gitterpunkte abgespeichert werden. Über NewPlot und **Disp** wird die Anzeige des Gitterpunktes erreicht.



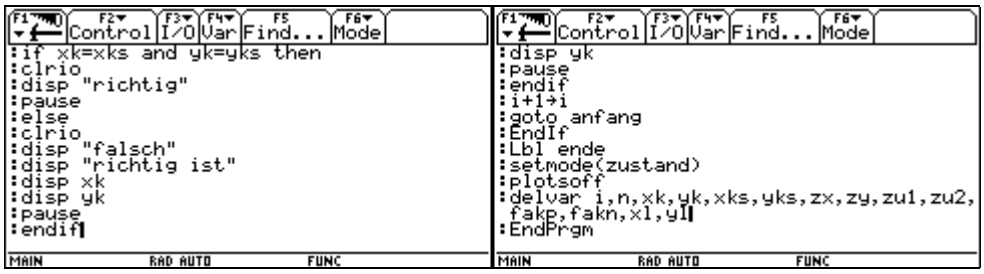
Die If-Abfrage muß noch beendet werden und dann testen wir unsere bisherige Programmierung:



Es folgen die zwei Dialogboxen für die Eingabe der Koordinaten.



Anschließende wird die Richtigkeit der Eingaben überprüft. Der MODE muss zurückgestellt und der angelegte Plot deaktiviert werden. Die benutzten Variablen gehören gelöscht. Auch am Programmanfang müssen die Variablen beim Löschbefehl ergänzt werden.



```
:if xk=xks and yk=yks then
:clrio
:disp "richtig"
:pause
:else
:clrio
:disp "falsch"
:disp "richtig ist"
:disp xk
:disp yk
:pause
:endif
```

```
:disp yk
:pause
:endif
:i+1→i
:goto anfang
:EndIf
:Lbl ende
:setmode(zustand)
:plotsoff
:delvar i,n,xk,yk,xks,yks,zx,zy,zu1,zu2,
 fakp,fakn,xl,yl
:EndPrgm
```

Lehrinhalte: Bildschirmteilung, Zufallszahlen

Befehle: setMode, setGraph,rand, PlotsOff, FnOff, getMode, Lbl, Goto, Dialog-EndDlog, Title, Text, dim, expr, local,If-Then-Else-EndIf, NewPlot, Disp, DispG, Pause, expr, ClrIo

Übung: Übungsprogramm zum Ablesen des Anstiegs eines Ursprungsgeraden

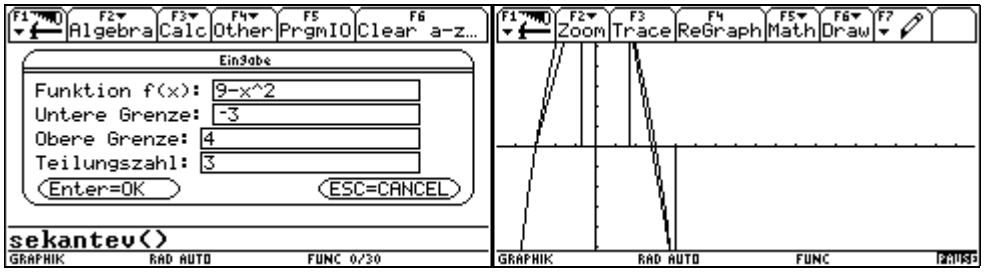
2.8 Programmierung auflösungsunabhängiger Graphiken

Programm: Darstellung des Sekantenverfahrens

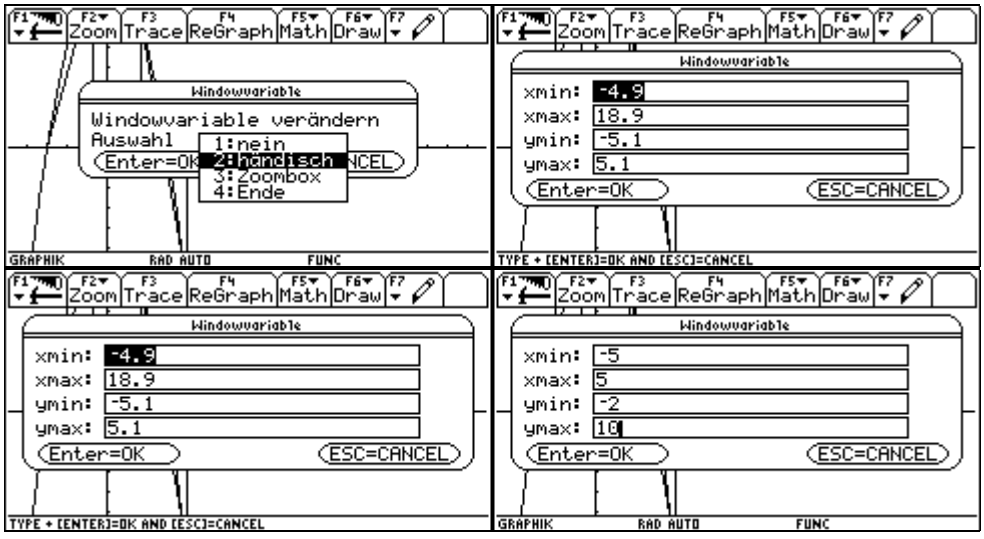
Benutzung:

Nach dem Programmstart wird man aufgefordert, die Funktion, das Intervall und die Teilungszahl festzulegen.

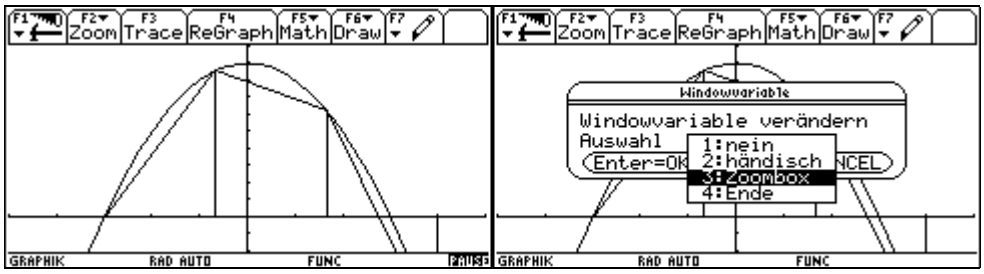
Danach erscheint die Graphik. Nach Beendigung der Pause gibt eine weitere Dialogbox die Möglichkeit die Graphikeinstellung zu ändern.



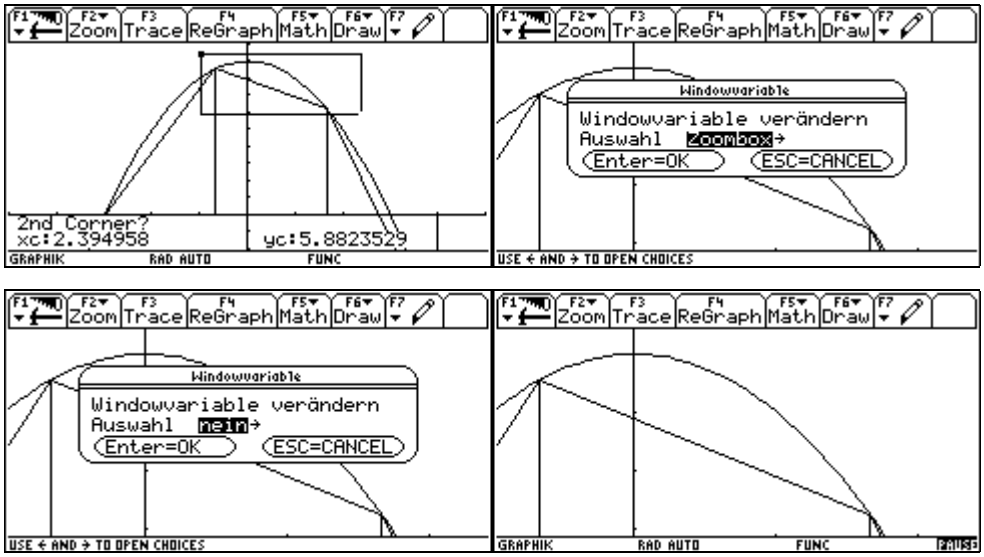
Wir wählen zunächst „händisch“. Eine weitere Dialogbox zeigt den momentanen Wert der [WINDOW]-Variablen und gibt Gelegenheit, diese entsprechend zu ändern.



Nach Ende der Pause wählen wir diesmal ZoomBox und können in gewohnter Weise einen neuen Ausschnitt festlegen.

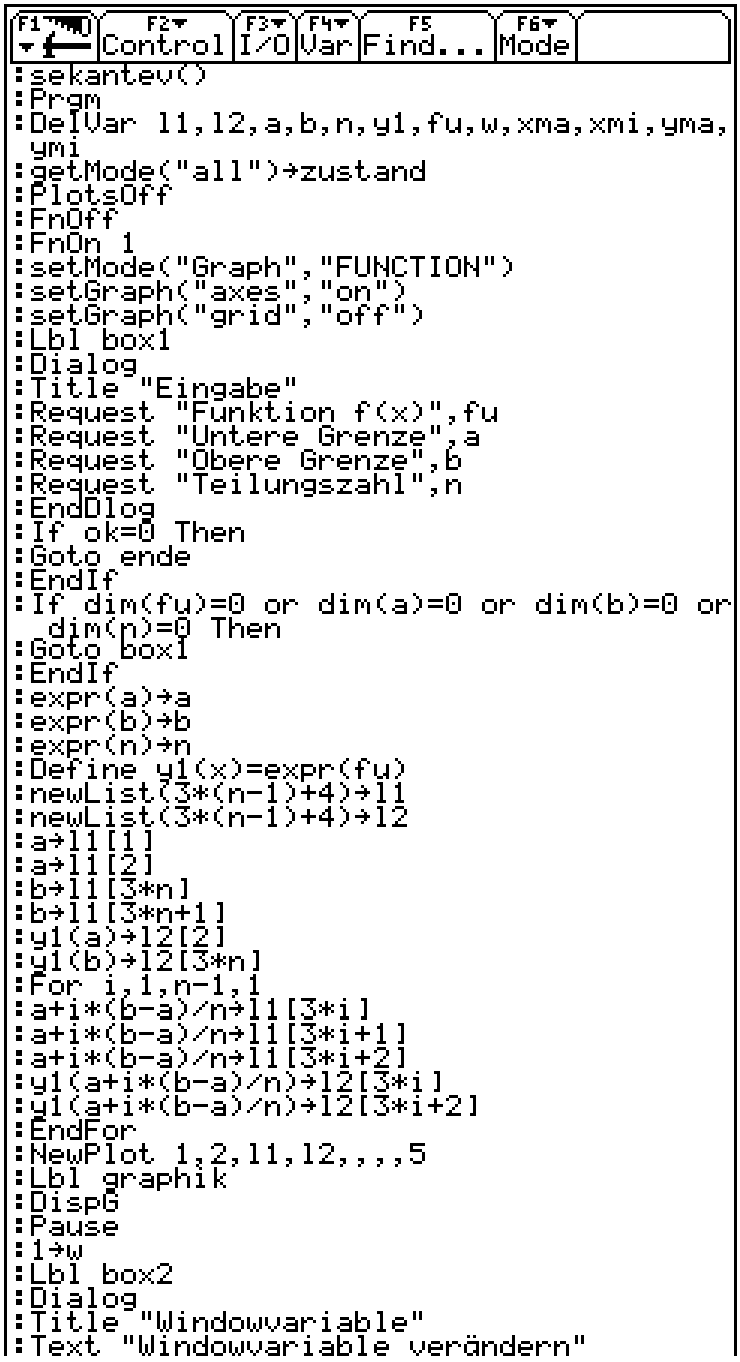


Nach der Pause wählen wir diesmal 1:nein und lassen damit die Graphik unverändert.



Nach dieser Pause wählen wir 4:Ende.

Programmcode:

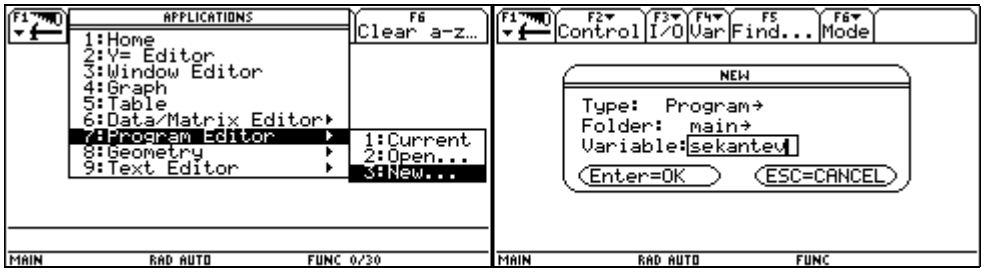


```
F1 F2 Control F3 I/O F4 Var F5 Find... F6 Mode
:sekantev()
:Prgm
:DelVar l1,l2,a,b,n,y1,fu,w,xma,xmi,yma,
 ymi
:getMode("all")→zustand
:PlotsOff
:FnOff
:FnOn 1
:setMode("Graph","FUNCTION")
:setGraph("axes","on")
:setGraph("grid","off")
:Lbl box1
:Dialog
:Title "Eingabe"
:Request "Funktion f(x)",fu
:Request "Untere Grenze",a
:Request "Obere Grenze",b
:Request "Teilungszahl",n
:EndDlog
:If ok=0 Then
:Goto ende
:EndIf
:If dim(fu)=0 or dim(a)=0 or dim(b)=0 or
 dim(n)=0 Then
:Goto box1
:EndIf
:expr(a)→a
:expr(b)→b
:expr(n)→n
:Define y1(x)=expr(fu)
:newList(3*(n-1)+4)→l1
:newList(3*(n-1)+4)→l2
:a→l1[1]
:a→l1[2]
:b→l1[3*n]
:b→l1[3*n+1]
:y1(a)→l2[2]
:y1(b)→l2[3*n]
:For i,1,n-1,1
:a+i*(b-a)/n→l1[3*i]
:a+i*(b-a)/n→l1[3*i+1]
:a+i*(b-a)/n→l1[3*i+2]
:y1(a+i*(b-a)/n)→l2[3*i]
:y1(a+i*(b-a)/n)→l2[3*i+2]
:EndFor
:NewPlot 1,2,l1,l2,,,,5
:Lbl graphik
:DispG
:Pause
:1→w
:Lbl box2
:Dialog
:Title "Windowvariable"
:Text "Windowvariable verändern"
```

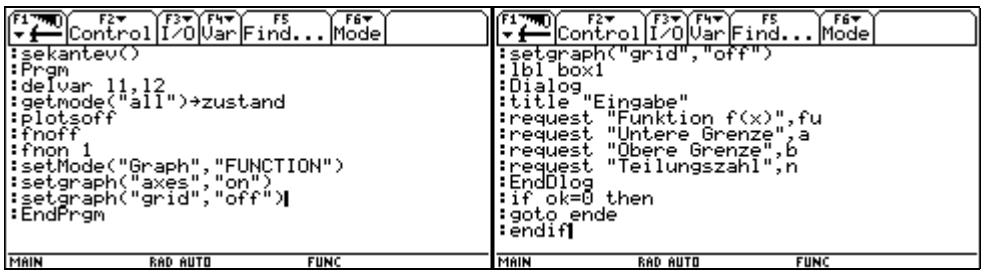
```
:DropDown "Auswahl",{"nein","händisch","
 Zoombox","Ende"},w
:EndDlog
:If ok=0 Then
:Goto ende
:EndIf
:If w=4 Then
:Goto ende
:EndIf
:If w=1 Then
:Goto graphik
:EndIf
:If w=3 Then
:ZoomBox
:Goto box2
:EndIf
:If w=2 Then
:string(xmax)→xma
:string(xmin)→xmi
:string(ymax)→yma
:string(ymin)→ymi
:Lbl box3
:Dialog
:Title "Windowvariable"
:Request "xmin",xmi
:Request "xmax",xma
:Request "ymin",ymi
:Request "ymax",yma
:EndDlog
:If ok=0 Then
:Goto ende
:EndIf
:If dim(xma)=0 or dim(xmi)=0 or dim(yma)
 =0 or dim(ymi)=0 Then
:Goto box3
:EndIf
:expr(xma)→xmax
:expr(xmi)→xmin
:expr(yma)→ymax
:expr(ymi)→ymin
:Goto graphik
:EndIf
:Lbl ende
:DelVar l1,l2,a,b,n,y1,w,i,xmi,xma,ymi,y
 ma,fu
:setMode(zustand)
:EndPrgm
```

Erstellen des Programms

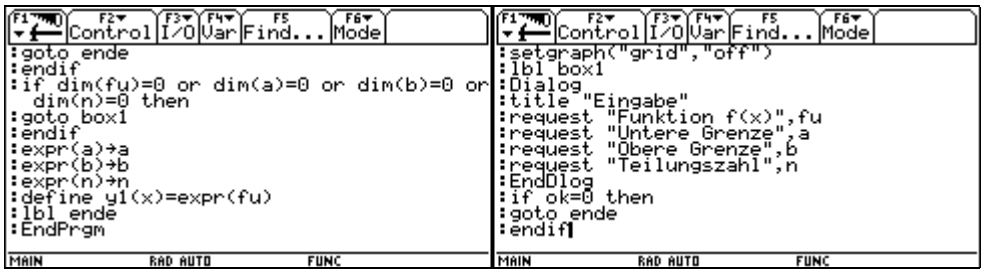
Wir öffnen de Programmeditor und geben dem neuen Programm den Namen sekantev.



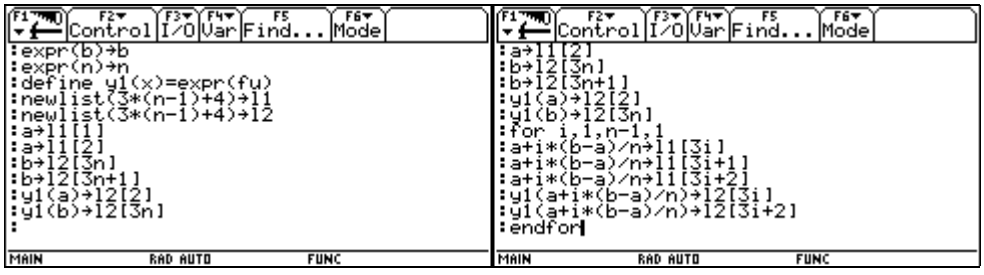
Nach dem Erstellen des üblichen Programmanfanges und dem Festlegen der Modi und der Graphik erzeugen wir eine Dialogbox für die Eingabe.



Hinter der Dialogbox werden ihre String-Eingaben in Variable umgewandelt und die Funktion als y1(x) gespeichert.

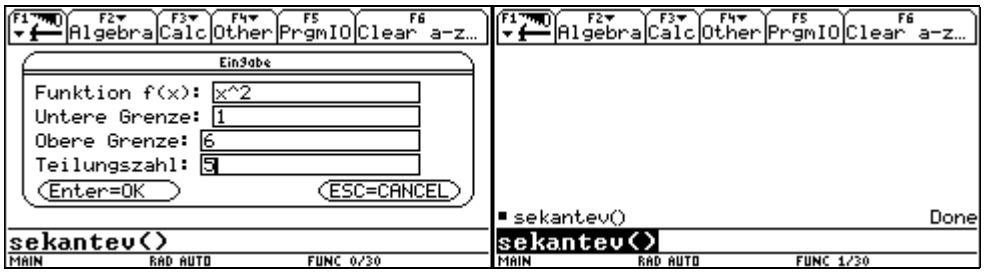


Dass die Zeichnung des Sekantenverfahrens von der Wahl der [WINDOW]-Variablen unabhängig ist, wird die gesamte Zeichnung als ein einziger Linienzug geplottet. Dazu muß eine Liste l1 für die *x*-Koordinaten und eine Liste l2 für die *y*-Koordinaten der entsprechenden Punkte erstellt werden. Die Erklärung für die folgenden Befehle wollen wir vorerst ein wenig aufschieben.

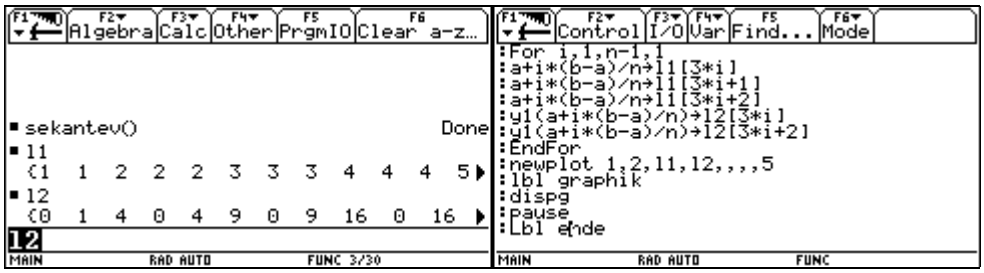


Wir starten nun das Programm, wählen folgende Eingaben und verlassen mit [ENTER] das Programm wieder. Nun betrachten wir im Homebereich die beiden Listen l1 und l2. Der Linienzug läuft beginnend vom Intervallanfang, zum Funktionswert des

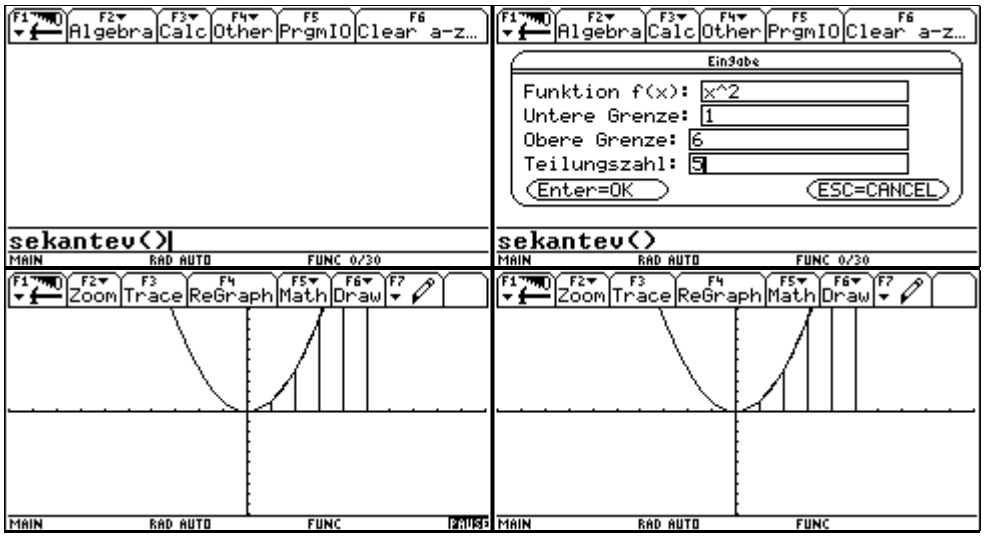
Intervallanfanges, zum Funktionswert des 1. Teilungspunktes, zum 1. Teilungspunkt, zum Funktionswert des 1. Teilungspunktes, zum Funktionswert des 2. Teilungspunktes ..... zum Funktionswert des Intervallendes und zum Intervallende.



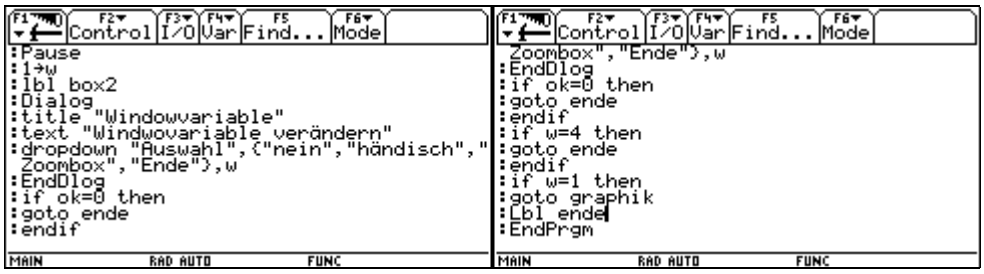
Über NewPlot erreichen wir das Plotten dieses Linienzuges. Bevor die Graphik über DispG angezeigt wird, setzen wir noch eine Sprungmarke graphik, zu der verzweigt wird, wenn die Graphikeinstellungen verändert werden.



Jetzt können wir unseren Plot ausprobieren. (Beachte die Syntax von NewPlot!)



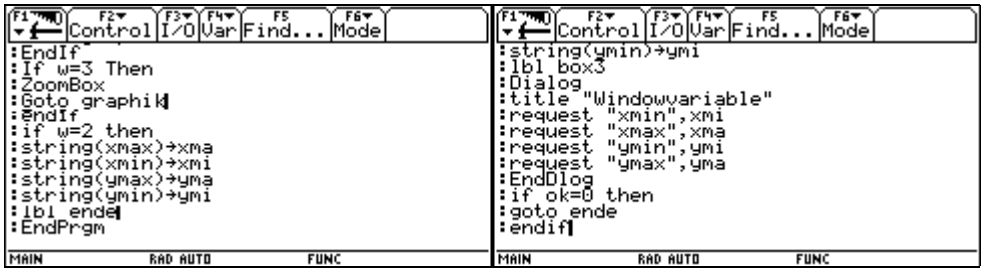
Es folgt die Erstellung der Dialogbox zum Verändern der [WINDOW]-Variablen. Im Falle, dass Ende gewählt wurde, muss nur ein Sprungbefehl zum Label ende gesetzt werden, im Falle, dass nein gewählt wurde, muss ein Sprungbefehl zum Label graphik gesetzt werden.



```
:Pause
:1→w
:lbl box2
:Dialog
:title "Windowvariable"
:text "Windowvariable verändern"
:dropdown "Auswahl",{"nein","händisch","
 Zoombox","Ende"},w
:EndDlog
:if ok=0 then
:goto ende
:endif
```

```
 Zoombox","Ende"},w
:EndDlog
:if ok=0 then
:goto ende
:endif
:if w=4 then
:goto ende
:endif
:if w=1 then
:goto graphik
:Lbl ende
:EndPrgm
```

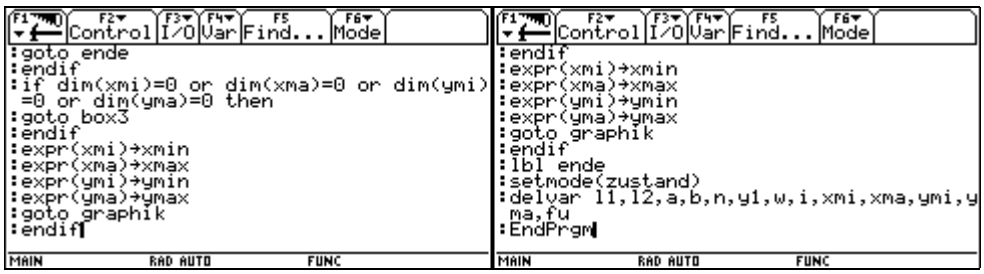
Wurde Zoombox gewählt, so benötigen wir nur den Befehl **ZoomBox** und danach einen Sprung zum Label graphik. Im Falle des händischen Veränderns der Fenstervariablen, wandeln wir die **aktuellen Werte in Strings** um und zeigen sie über Request in einer Dialogbox an.



```
:EndIf
:If w=3 Then
:ZoomBox
:Goto graphik
:EndIf
:if w=2 then
:string(xmax)→xma
:string(xmin)→xmi
:string(ymax)→yma
:string(ymin)→ymi
:lbl ende
:EndPrgm
```

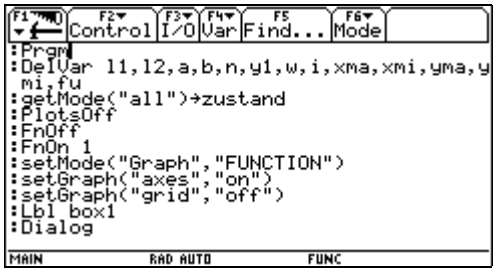
```
:string(ymin)→ymi
:lbl box3
:Dialog
:title "Windowvariable"
:request "xmin",xmi
:request "xmax",xma
:request "ymin",ymi
:request "ymax",yma
:EndDlog
:if ok=0 then
:goto ende
:endif
```

Nach dem Verlassen der Dialogbox müssen die Strings wieder in Werte umgewandelt werden, wodurch auch die [WINDOW]-Variablen mit neuen Werten belegt werden. Zuletzt wird zum Label graphik zurückgesprungen. Am Ende des Programms müssen wir noch die Modi zurückstellen und die Variablen löschen. Das Löschen muss - wie immer - auch noch zu Beginn des Programms vervollständigt werden.



```
:goto ende
:endif
:if dim(xmi)=0 or dim(xma)=0 or dim(ymi)
 =0 or dim(yma)=0 then
:goto box3
:endif
:expr(xmi)→xmin
:expr(xma)→xmax
:expr(ymi)→ymin
:expr(yma)→ymax
:goto graphik
:endif
```

```
:endif
:expr(xmi)→xmin
:expr(xma)→xmax
:expr(ymi)→ymin
:expr(yma)→ymax
:goto graphik
:endif
:lbl ende
:setmode(zustand)
:delvar l1,l2,a,b,n,y1,w,i,xmi,xma,ymi,y
 ma,fu
:EndPrgm
```



```
:Prgm
:DelVar l1,l2,a,b,n,y1,w,i,xma,xmi,yma,y
 mi,fu
:getMode("all")→zustand
:PlotsOff
:FnOff
:FnOn 1
:setMode("Graph","FUNCTION")
:setGraph("axes","on")
:setGraph("grid","off")
:Lbl box1
:Dialog
```

Lehrinhalte: Steuerung der [WINDOW]-Variablen, Zoomboxen über Programme, Erstellen von Graphiken mit Hilfe der Plotfunktion

Befehle: DelVar, getMode, setMode, setGraph, PlotsOff, FnOff, Lbl, Goto, Dialog-EndDlog, Title, Text, Request, dim, expr, Define, newList, For-EndFor, DispG, Pause, ZoomBox, If-Then-EndIf, NewPlot, xmin, xmax, ymin, ymax