Introduction to TI-Nspire™ Programming Editor

Weitere Informationen zu TI Technology finden Sie in der Online-Hilfe unter education.ti.com/eguide.

Wichtige Informationen

Soweit in der Lizenz, die ein Programm begleitet, ausdrücklich anders angegeben ist, übernimmt Texas Instruments keine ausdrückliche oder stillschweigende Gewährleistung, einschließlich, aber nicht beschränkt auf implizierte Garantien der Marktgängigkeit und Eignung für einen bestimmten Zweck, in Bezug auf Programme oder Buchmaterialien und macht dies Materialien, die ausschließlich auf einer "as-is" - Basis zur Verfügung stehen. In keinem Fall haftet Texas Instruments für besondere, unsachgemäße oder zufällige Schäden, die im Zusammenhang mit dem Kauf oder der Verwendung dieser Materialien entstehen, und die alleinige und ausschließliche Haftung von Texas Instruments, unabhängig von der Form von Aktion darf den in der Lizenz für das Programm festgelegten Betrag nicht überschreiten. Darüber hinaus haftet Texas Instruments nicht für jegliche Ansprüche jeglicher Art gegen die Verwendung dieser Materialien durch eine andere Partei.

Marken und Urheberrechte

Die TI-Nspire ™ Software nutzt Lua als Scripting-Umgebung. Für Urheberrechts- und Lizenzinformationen siehe http://www.lua.org/license.html.

Die TI-Nspire ™ Software nutzt die Chipmunk Physics Version 5.3.4 als Simulationsumgebung. Für Lizenzinformationen siehe http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/.

Microsoft® und Windows® sind eingetragene Marken der Microsoft Corporation in den USA und / oder anderen Ländern.

Mac OS®, iPad® und OS X® sind eingetragene Warenzeichen der Apple Inc.

Unicode® ist ein eingetragenes Warenzeichen von Unicode, Inc. in den USA und anderen Ländern.

Bluetooth® Wortmarke und Logos sind eingetragene Warenzeichen von Bluetooth SIG, Inc.

Contents

Erste Schritte mit dem Programmeditor

Sie können benutzerdefinierte Funktionen oder Programme erstellen, indem Sie Definitionsanweisungen in die Calculator-Eingabezeile eingeben oder den Programmeditor verwenden. Der Programmeditor bietet einige Vorteile und ist daher Gegenstand dieses Abschnitts. Weitere Informationen finden Sie unter *Calculator*.

- Der Editor ist mit Programmiervorlagen und Dialogfeldern ausgestattet, die Ihnen helfen, Funktionen und Programme mit der korrekten Syntax zu definieren.
- Der Editor ermöglicht die Eingabe mehrzeiliger Programmanweisungen, ohne dass eine spezielle Tastenfolge zum Hinzufügen der einzelnen Zeilen erforderlich ist.
- Sie können bequem private und öffentliche Bibliotheksobjekte (Variablen, Funktionen und Programme) erstellen. Weitere Informationen finden Sie unter *Bibliotheken*.

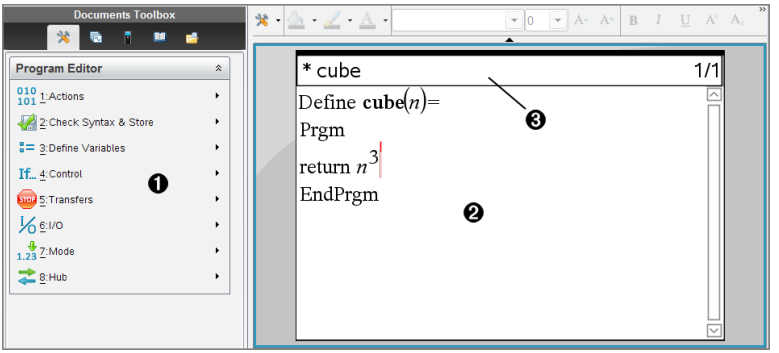
Starten des Programmeditors

▸ So fügen Sie dem aktuellen Problem eine neue Programmeditor-Seite hinzu:

Klicken Sie auf der Symbolleiste auf **Einfügen > Programmeditor > Neu**.

Handheld: Drücken Sie doc und wählen Sie dann **Einfügen > Programmeditor > Neu** aus.

Hinweis: Den Editor können Sie auch über das Menü **Funktionen & Programme** der neuen Calculator-Seite aufrufen.



❶ Programmeditor-Menü – Dieses Menü steht Ihnen in der Normalansicht im Programmeditor-Arbeitsbereich jederzeit zur Verfügung.

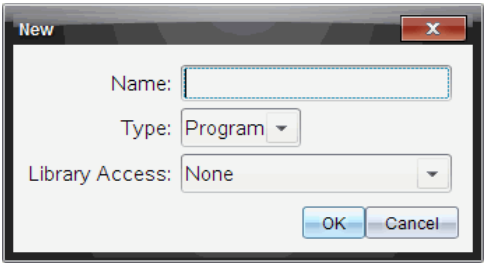
❷ Programmeditor-Arbeitsbereich

❸ Statuszeile gibt die Zeilennummer und den Namen der Funktion oder des Programms an, die/das gerade bearbeitet wird. Ein Sternchen (*) zeigt an, dass diese Funktion „schmutzig" ist, was bedeutet, dass die Funktion seit der letzten Syntaxüberprüfung und Speicherung verändert wurde.

Definieren eines Programms oder einer Funktion

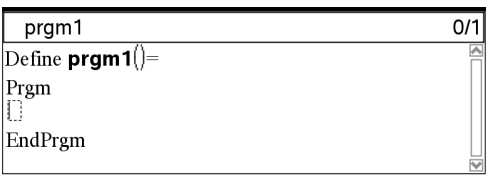
Einen neuen Programmeditor starten

1. Um den Editor von einer Calculator-Seite zu starten:
   - Wählen Sie im Menü **Einfügen** den Punkt **Programmeditor und** anschließend **Neu**.
2. Um den Editor von außerhalb einer Calculator-Seite zu starten:
   - Wählen Sie im Menü **Einfügen** den Punkt **Programmeditor und** anschließend **Neu**.



3. Geben Sie einen Namen für die Funktion oder das Programm ein, die/das Sie definieren.
4. Wählen Sie den **Typ (Type)** aus (**Programm** oder **Funktion**).
5. Legen Sie den **Bibliothekszugriff (Library Access)** fest:
   - Wenn Sie die Funktion oder das Programm nur aus dem aktuellen Dokument und Problem heraus nutzen möchten, wählen Sie **Kein (None)**.
   - Wenn Sie von jedem beliebigen Dokument auf die Funktion oder das Programm zugreifen können möchten, jedoch nicht möchten, dass die Funktion / das Programm im Katalog aufgeführt wird, wählen Sie **LibPriv**.
   - Wenn Sie möchten, dass die Funktion oder das Programm aus jedem Dokument heraus aufgerufen werden kann und im Katalog angezeigt wird, wählen Sie **LibPub (Im Katalog zeigen / Show in Catalog)**. Einzelheiten finden Sie im Kapitel "*Bibliotheken*".
6. Klicken Sie auf **OK**.

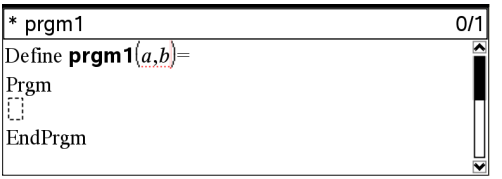
   Eine neue Instanz des Programmeditors wird geöffnet, in der eine Vorlage angezeigt wird, die die von Ihnen gewählten Optionen erfüllt.



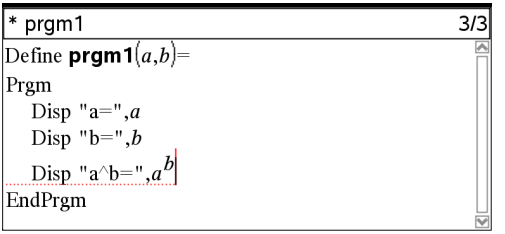
Eingeben von Zeilen in eine Funktion oder ein Programm

Der Programmeditor führt bei der Eingabe keine Befehle aus oder evaluiert Ausdrücke. Befehle werden erst ausgeführt, wenn Sie die Funktion auswerten oder das Programm ausführen.

1. Wenn Ihre Funktion oder Ihr Programm den Anwender auffordert, Argumente einzugeben, geben Sie Parameternamen in die Klammern hinter dem Namen ein. Trennen Sie Parameter mit einem Komma.



2. Geben Sie zwischen den Zeilen Func und EndFunc (oder Prgm und EndPrgm) die Anweisungen ein, aus denen Ihre Funktion bzw. Ihr Programm besteht.



- Sie können entweder die Namen von Funktionen und Befehlen eingeben oder diese aus dem Katalog auswählen.
- Eine Zeile kann länger sein als der Bildschirm breit ist; in diesem Fall müssen Sie im Fensterbereich scrollen, um die vollständige Anweisung zu sehen.
- Drücken Sie nach jeder Zeile die Taste enter. Dadurch wird eine neue leere Zeile eingefügt, in der Sie die Eingabe fortsetzen können.
- Verwenden Sie die Pfeiltasten ◀, ▶, ▲ und ▼, um zur Eingabe oder Bearbeitung von Befehlen durch die Funktion oder das Programm zu scrollen.

Einfügen von Kommentaren

Über ein Kommentarsymbol (©) können Sie Bemerkungen einfügen. Kommentare können für Nutzer hilfreich sein, die ein Programm ansehen oder bearbeiten. Kommentare werden nicht angezeigt, wenn das Programm ausgeführt wird, und haben keinen Einfluss auf den Programmablauf.

```
Define LibPub volcyl(ht,r) =
Prgm
©volcyl(ht,r) => Zylindervolumen ❶
  Disp "Volume =", approx(π • r^2 • ht)
```

```
©Dies ist ein weiterer Kommentar.
EndPrgm
```

❶ Kommentar, der die benötigte Syntax anzeigt. Da dies ein öffentliches Bibliotheksobjekt ist und dieser Befehl in der ersten Zeile eines Func- oder Prgm-Blocks steht, wird der Kommentar im Katalog als Hilfe angezeigt. Einzelheiten finden Sie im Kapitel "*Bibliotheken*".

So fügen Sie einen Kommentar ein:

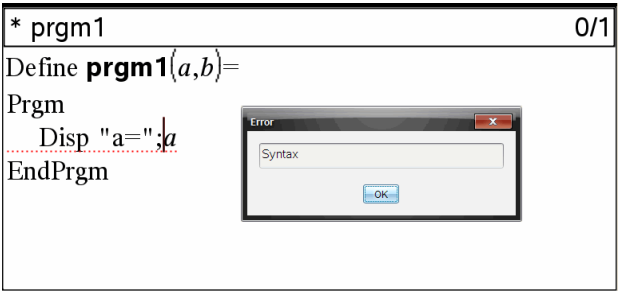
1. Platzieren Sie den Cursor am Ende der Zeile, in die Sie einen Kommentar einfügen möchten.
2. Wählen Sie im Menü **Aktionen** den Befehl **Kommentar einfügen**.
3. Geben Sie hinter dem Symbol © den Kommentartext ein.

Überprüfen der Syntax

Mit dem Programmeditor können Sie Funktionen und Programme auf korrekte Syntax überprüfen.

- Wählen Sie im Menü **Syntax prüfen & speichern** den Punkt **Syntax prüfen**.

  Wenn bei der Syntaxüberprüfung Syntaxfehler festgestellt werden, wird eine Fehlermeldung angezeigt und der Cursor neben dem ersten Fehler platziert, damit Sie ihn korrigieren können.



Speichern der Funktion / des Programms

Um einen Zugriff auf Ihre Funktion oder Ihr Programm zu ermöglichen, müssen Sie diese(s) speichern. Der Programmeditor überprüft vor dem Speichern automatisch die Syntax.

In der oberen linken Ecke des Programmeditors wird ein Sternchen (*) angezeigt, um darauf hinzuweisen, dass die Funktion bzw. das Programm noch nicht gespeichert wurde.

- Wählen Sie im Menü **Syntax prüfen & speichern** den Punkt **Syntax prüfen & speichern**.

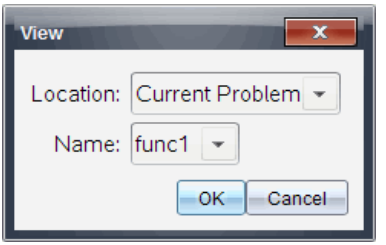
Wenn bei der Syntaxüberprüfung Syntaxfehler festgestellt werden, wird eine Fehlermeldung angezeigt und versucht, den Cursor nahe dem ersten Fehler zu platzieren.

Wenn keine Syntaxfehler gefunden werden, wird die Meldung “Erfolgreich gespeichert” in der Statuszeile oben im Programmeditor angezeigt .

Hinweis: Wenn die Funktion bzw. das Programm als Bibliotheksobjekt definiert wurde, müssen Sie das Dokument im zugewiesenen Bibliotheksordner speichern und die Bibliotheken aktualisieren, um das Objekt anderen Dokumenten zur Verfügung zu stellen. Einzelheiten finden Sie im Kapitel “*Bibliotheken*”.

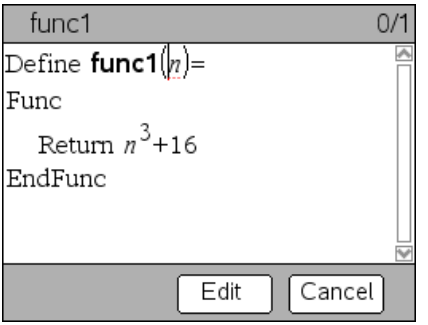
Anzeigen eines Programms oder einer Funktion

1. Wählen Sie im Menü **Aktionen** den Punkt **Ansicht**.



2. Wenn die Funktion / das Programm ein Bibliotheksobjekt ist, wählen Sie die entsprechende Bibliothek in der Liste **Speicherort (Location)**.
3. Wählen Sie den Namen der Funktion / des Programms in der Liste **Name**.

   Die Funktion bzw. das Programm wird in einem Anzeigefenster geöffnet.



4. Verwenden Sie zum Anzeigen der Funktion / des Programms die Pfeiltasten.
5. Wenn Sie das Programm bearbeiten möchten, klicken Sie auf **Bearbeiten**.

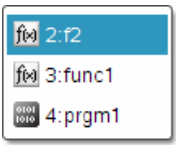
   Hinweis: Der Menüpunkt **Bearbeiten (Edit)** ist nur für Funktionen und Programme verfügbar, die im aktuellen Problem definiert wurden. Um ein Bibliotheksobjekt zu bearbeiten, müssen Sie zuerst das zugehörige Bibliotheksdokument öffnen.

Öffnen einer Funktion oder eines Programms zum Bearbeiten

Sie können Funktionen oder Programme nur aus dem aktuellen Problem heraus öffnen.

Hinweis: Gesperrte Programme oder Funktionen können nicht geändert werden. Um das Objekt zu entsperren, wechseln Sie auf eine Calculator-Seite und benutzen Sie den Befehl **Entsperren (unLock)**.

1. Zeigen Sie die Liste der verfügbaren Funktionen und Programme an.
   - Wählen Sie im Menü **Aktionen** den Punkt **Öffnen**.

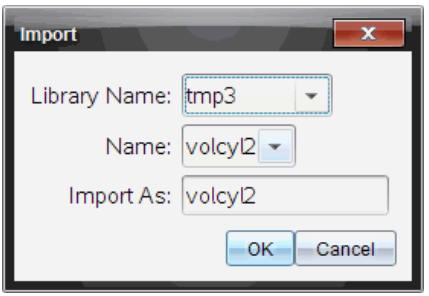


2. Wählen Sie das Objekt, das Sie öffnen möchten.

Importieren eines Programms aus der Bibliothek

Sie können eine als Bibliotheksobjekt definierte Funktion oder ein entsprechendes Programm in einen Programmeditor innerhalb des aktuellen Problems importieren. Die importierte Kopie ist nicht gesperrt, auch wenn das Original gesperrt ist.

1. Wählen Sie im Menü **Aktionen** den Punkt **Importieren**.



2. Wählen Sie den **Bibliotheksnamen (Library Name)**.
3. Wählen Sie den **Namen (Name)** des Objekts.
4. Wenn Sie dem importierten Objekt einen anderen Namen geben möchten, geben Sie den Namen in das Feld **Importieren als (Import As)** ein.

Erstellen einer Kopie von einer Funktion / einem Programm

Wenn Sie eine neue Funktion bzw. ein neues Programm erstellen möchten, können Sie mit einer Kopie der aktuellen Funktion / des aktuellen Programms als Vorlage beginnen. Die Kopie, die Sie erstellen, ist nicht gesperrt, auch wenn das Original gesperrt ist.

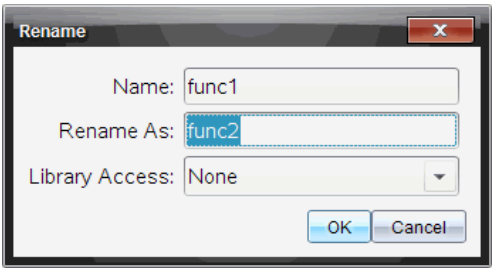
1. Wählen Sie im Menü **Aktionen** den Punkt **Kopie erstellen**.

2. Geben Sie einen neuen Namen ein oder klicken Sie auf **OK**, um den vorgeschlagenen Namen zu akzeptieren.
3. Um die Zugriffsebene zu ändern, wählen Sie **Bibliothekszugriff (Library Access)** und wählen Sie die neue Ebene.

Umbenennen eines Programms / einer Funktion

Sie können die aktuelle Funktion / das aktuelle Programm umbenennen und (optional) deren/dessen Zugriffsebene ändern.

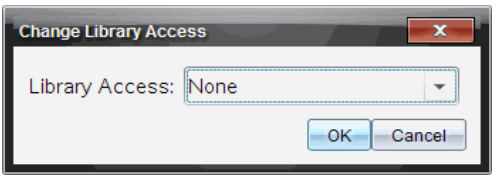
1. Wählen Sie im Menü **Aktionen** den Punkt **Umbenennen**.



2. Geben Sie einen neuen Namen ein oder klicken Sie auf **OK**, um den vorgeschlagenen Namen zu akzeptieren.
3. Wenn Sie die Zugriffsebene ändern möchten, wählen Sie **Bibliothekszugriff (Library Access)** und dann eine neue Ebene.

Ändern der Bibliothekszugriffsebene

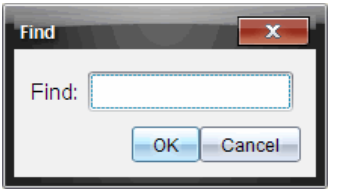
1. Wählen Sie im Menü **Aktionen** den Punkt **Bibliothekszugriff ändern**.



2. Wählen Sie den **Bibliothekszugriff (Library Access)**:
   - Wenn Sie die Funktion / das Programm nur aus dem aktuellen Calculator-Problem heraus nutzen möchten, wählen Sie **Kein (None)**.
   - Wenn Sie von jedem beliebigen Dokument auf die Funktion oder das Programm zugreifen können möchten, jedoch nicht möchten, dass die Funktion / das Programm im Katalog aufgeführt wird, wählen Sie **LibPriv**.
   - Wenn Sie die Funktion / das Programm für jedes Dokument erreichbar machen und außerdem im Katalog sichtbar machen möchten, wählen Sie **LibPub**.

Text suchen

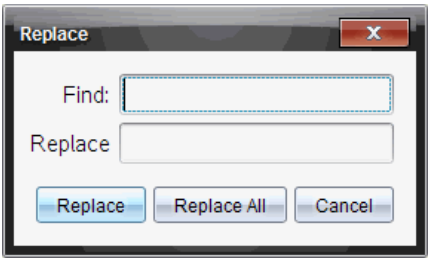
1. Wählen Sie im Menü **Aktionen** den Punkt **Suchen**.



2. Geben Sie den zu suchenden Text ein und klicken Sie auf **OK.**
   - Wenn der Text gefunden wird, wird er im Programm markiert.
   - Wenn der Text nicht gefunden wird, wird eine entsprechende Meldung angezeigt.

Suchen und Ersetzen von Text

1. Wählen Sie im Menü **Aktionen** den Punkt **Suchen und ersetzen**.



2. Geben Sie den Text ein, den Sie suchen.
3. Geben Sie den Text ein, der den gesuchten Text ersetzen soll.
4. Klicken Sie auf **Ersetzen**, um die erste Textstelle hinter der Cursorposition zu ersetzen, oder klicken Sie auf **Alle ersetzen**, um alle entsprechenden Textstellen zu ersetzen.

   Hinweis: Wenn der Text in einer mathematischen Vorlage gefunden wird, wird eine Warnmeldung angezeigt, die Sie darüber informiert, dass der Ersatztext die gesamte Vorlage ersetzt und nicht nur die gefundene Textstelle.

Schließen der aktuellen Funktion / des aktuellen Programms

▸ Wählen Sie im Menü **Aktionen** den Punkt **Schließen**.

Wenn die Funktion / das Programm ungespeicherte Änderungen enthält, werden Sie aufgefordert, vor dem Schließen die Syntax zu überprüfen und die Änderungen zu speichern.

Ausführen von Programmen und Auswerten von Funktionen

Nachdem Sie ein Programm bzw. eine Funktion definiert und gespeichert haben, können Sie es/sie aus einer Anwendung heraus verwenden. Alle Applikationen können

Funktionen auswerten, aber nur die Applikationen Calculator und Notes können Programme ausführen.

Die Anweisungen eines Programms werden der Reihe nach ausgeführt (obwohl einige Befehle den Programmablauf ändern). Die Ausgabe – soweit vorhanden – erfolgt im Arbeitsbereich der Applikation.

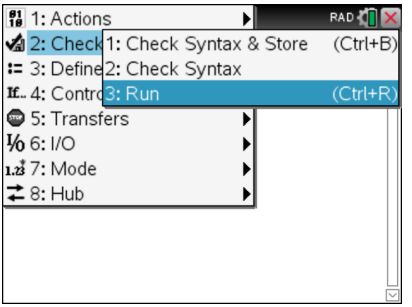
- Das Programm wird ausgeführt, bis die letzte Anweisung oder ein **Stopp** (Stop)-Befehl erreicht wird.
- Eine Funktion wird ausgeführt, bis ein Zurück **Zurück** (Return)-Befehl erreicht wird.

Ausführen eines Programms bzw. einer Funktion im Programmeditor

1. Stellen Sie sicher, dass Sie ein Programm bzw. eine Funktion definiert haben und der Programmeditor der aktive Fensterbereich (Computer) oder die aktive Seite (Handheld) ist.

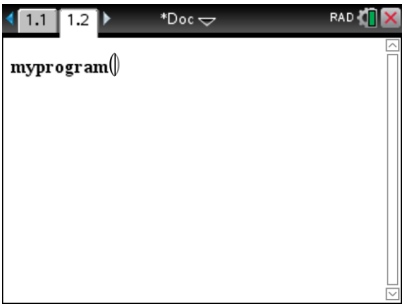
2. Klicken Sie in der Symbolleiste auf die Taste **Dokumente-Werkzeuge** und wählen Sie **Syntax prüfen und speichern > Ausführen**.
   – oder –
   Drücken Sie **Strg+R**.

   Handheld: Drücken Sie menu 2 3 oder drücken Sie ctrl R.



   Hierdurch wird automatisch:

   - die Syntax überprüft und das Programm bzw. die Funktion gespeichert;
   - der Name des Programms bzw. der Funktion in der ersten verfügbaren Zeile der Calculator Applikation direkt hinter dem Programmeditor eingefügt. Wenn kein Calculator in dieser Position vorhanden ist, wird ein neuer eingefügt.



3. Wenn das Programm bzw. die Funktion Sie auffordert, Argumente anzugeben, geben Sie die Werte oder Variablennamen zwischen den Klammern ein.
4. Drücken Sie enter.

Hinweis: Sie können auch ein Programm bzw. eine Funktion in der Applikation Calculator oder Notes ausführen, indem Sie den Namen des Programms mit Klammern und alle erforderlichen Argumente eingeben und dann auf enter drücken.

Verwendung von Kurz- und Langnamen

Jedes Mal, wenn im gleichen Problem ein Objekt definiert ist, können Sie auf dieses zugreifen, indem Sie seinen Kurznamen eingeben (den Namen, der im Befehl **Definieren** (Define) angegeben ist). Dies gilt für alle definierten Objekte einschließlich privater und öffentlicher Objekte sowie Objekte, die nicht in Bibliotheken enthalten sind.

Ein Bibliotheksobjekt können Sie aus jedem Dokument aufrufen, indem Sie den Langnamen des Objekts eingeben. Ein Langname besteht aus dem Namen des Bibliotheksdokuments des Objekts, gefolgt von einem Backslash „\“ und dem Namen des Objekts. So ist zum Beispiel der Langname des Objekts, das im Bibliotheksdokument **lib1** als **func1** definiert ist, **lib1\func1**. Um das Zeichen „\“ am Handheld einzugeben, drücken Sie shift ÷.

Hinweis: Wenn Sie den genauen Namen oder die Reihenfolge der Argumente für ein Objekt aus einer privaten Bibliothek nicht mehr wissen, können Sie das Bibliotheksdokument öffnen oder den Programmeditor verwenden, um das Objekt anzuzeigen. Außerdem können Sie sich mit **getVarInfo** eine Liste der Objekte in einer Bibliothek anzeigen lassen.

Verwenden eines Programms bzw. einer Funktion aus einer öffentlichen Bibliothek

1. Stellen Sie sicher, dass Sie das Objekt im ersten Problem des Dokuments definiert haben, das Objekt gespeichert haben, das Bibliotheksdokument im Ordner MyLib gespeichert haben und die Bibliotheken aktualisiert haben.
2. Öffnen Sie die TI-Nspire™-Applikation, in der Sie das Programm bzw. die Funktion verwenden möchten.

   Hinweis: Alle Applikationen können Funktionen auswerten, aber nur die Applikationen Calculator und Notes können Programme ausführen.

3. Öffnen Sie den Katalog und verwenden Sie die Bibliotheksregisterkarte, um das Objekt zu suchen und einzufügen.
   – oder –
   Geben Sie den Namen des Objekts ein. Hängen Sie bei Programmen oder Funktionen immer Klammern an den Namen an.

   ```
   libs2\func1()
   ```

4. Wenn das Programm bzw. die Funktion Sie auffordert, Argumente anzugeben, geben Sie die Werte oder Variablennamen zwischen den Klammern ein.

   ```
   libs2\func1(34,Power)
   ```

5. Drücken Sie enter.

Verwenden eines privaten Bibliotheksprogramms bzw. einer privaten Bibliotheksfunktion

Um ein Objekt aus einer privaten Bibliothek zu verwenden, müssen Sie dessen Langnamen kennen. Der Langname des Objekts, das im Bibliotheksdokument **lib1** als **func1** definiert ist, lautet zum Beispiel **lib1\func1**.

Hinweis: Wenn Sie den genauen Namen oder die Reihenfolge der Argumente für ein Objekt aus einer privaten Bibliothek nicht mehr wissen, können Sie das Bibliotheksdokument öffnen oder den Programmeditor verwenden, um das Objekt anzuzeigen.

1. Stellen Sie sicher, dass Sie das Objekt im ersten Problem des Dokuments definiert haben, das Objekt gespeichert haben, das Bibliotheksdokument im Ordner MyLib gespeichert haben und die Bibliotheken aktualisiert haben.
2. Öffnen Sie die TI-Nspire™-Applikation, in der Sie das Programm bzw. die Funktion verwenden möchten.

   Hinweis: Alle Applikationen können Funktionen auswerten, aber nur die Applikationen Calculator und Notes können Programme ausführen.

3. Geben Sie den Namen des Objekts ein. Hängen Sie bei Programmen oder Funktionen immer Klammern an den Namen an.

   ```
   libs2\func1()
   ```

4. Wenn das Programm Sie auffordert, Argumente anzugeben, geben Sie die Werte oder Variablennamen zwischen den Klammern ein.

   ```
   libs2\func1(34,Power)
   ```

5. Drücken Sie enter.

Unterbrechen eines laufenden Programms bzw. einer laufenden Funktion

Während ein Programm oder eine Funktion ausgeführt wird, wird das Symbol 'Beschäftigt' ⧖ angezeigt.

- Um das Programm bzw. die Funktion abzubrechen:

- Windows®: Halten Sie die Taste **F12** gedrückt und drücken Sie mehrmals die **Eingabetaste**.
- Mac®: Halten Sie die Taste **F5** gedrückt und drücken Sie mehrmals die **Eingabetaste**.
- Handheld: Halten Sie die Taste [on] gedrückt und drücken Sie mehrmals [enter].

Es wird eine Meldung angezeigt. Zum Bearbeiten des Programms oder der Funktion im Programmeditor, wählen Sie **Gehe zu** (Go To). Der Cursor wird bei dem Befehl angezeigt, an dem der Programmstopp erfolgte.

Werte in ein Programm eingeben

Um die Werte einzugeben, die eine Funktion oder ein Programm für ihre bzw. seine Berechnungen verwendet, gibt es mehrere Möglichkeiten.

Einbetten der Werte im Programm / in der Funktion

Diese Methode eignet sich hauptsächlich für Werte, die bei jedem Aufruf des Programms bzw. der Funktion identisch sein müssen.

1. Definieren Sie das Programm.

```
Define calculatearea()=
Prgm
w:=3
h:=23.64
area:=w*h
EndPrgm
```

2. Starten Sie das Programm.

```
calculatearea() :area                    70.92
```

Zuweisen von Werten an Variable durch den Benutzer

Ein Programm bzw. eine Funktion kann auf zuvor erstellte Variablen verweisen. Bei dieser Methode muss sich der Benutzer die Variablennamen merken und ihnen vor dem Aufruf des betreffenden Objekts Werte zuweisen.

1. Definieren Sie das Programm.

```
Define calculatearea()=
Prgm
area:=w*h
EndPrgm
```

2. Weisen Sie die Werte zu und starten Sie das Programm.

```
w:=3 : h:=23.64
calculatearea() :area                    70.92
```

Übergeben von Werten als Argumente durch den Benutzer

Bei dieser Methode kann der Benutzer in dem Ausdruck, aus dem das Programm bzw. die Funktion aufgerufen wird, einen oder mehrere Werte als Argumente übergeben.

Das folgende Programm **volcyl** berechnet das Volumen eines Zylinders. Der Benutzer muss zwei Werte angeben: die Höhe und den Radius des Zylinders.

1. Definieren Sie das Programm **volcyl.**

```
Definevolcyl(height,radius) =
Prgm
Disp "Volumen =", approx(π • radius^2 • height)
EndPrgm
```

2. Führen Sie das Programm aus, um das Volumen eines Zylinders mit einer Höhe von 34 mm und einem Radius von 5 mm anzuzeigen.

```
volcyl(34,5)          Volumen = 534.071
```

Hinweis: Sie müssen beim Ausführen des Programms **volcyl** die Parameternamen nicht verwenden, jedoch müssen Sie die beiden Argumente eingeben (als Werte, Variablen oder Ausdrücke). Der erste muss die Höhe angeben, der zweite den Radius.

Interaktives Erfragen der Werte vom Benutzer (nur bei Programmen)

Mit den Befehlen **Request** und **RequestStr** können Sie in einem Programm den Ablauf unterbrechen und den Benutzer über ein Dialogfeld um die Eingabe von Daten bitten. So muss sich der Benutzer weder die Variablennamen merken noch die Reihenfolge, in der sie benötigt werden.

In Funktionen sind die Befehle **Request** und **RequestStr** nicht zulässig.

1. Definieren Sie das Programm.

```
Define calculatearea()=
Prgm
  Request "Breite: ",w
  Request "Höhe: ",h
area:=w*h
EndPrgm
```

2. Starten Sie das Programm und machen Sie bei den Abfragen entsprechende Eingaben.

```
calculatearea() : area
Breite: 3          (3 als Antwort eingegeben)
Höhe: 23.64  (23.64 als Antwort eingegeben)
                                        70.92
```

Verwenden Sie **RequestStr** anstelle von **Request**, wenn das Programm die Benutzereingabe als String und nicht als mathematischen Ausdruck interpretieren soll.

Auf diese Weise muss der Benutzer die Eingabe nicht in Anführungszeichen ("") setzen.

Anzeigen von Information

Laufende Funktionen und Programme zeigen errechnete Zwischenergebnisse nur an, wenn Sie einen entsprechenden Befehl einfügen. Dies ist ein wichtiger Unterschied zwischen einer Berechnung in der Eingabezeile und der Berechnung in einer Funktion bzw. einem Programm.

Die folgenden Berechnungen zeigen beispielsweise Ergebnisse in einer Funktion oder einem Programm nicht an (von der Eingabezeile aus hingegen schon).

```
⋮
x:=12•6
cos(π/4)➞
⋮
```

Anzeigen von Informationen aus dem Protokoll

Mit dem Befehl **Disp** können Sie in einem Programm oder einer Funktion Informationen (auch Zwischenergebnisse) aus dem Protokoll anzeigen.

```
⋮
Disp 12•6
Disp "Ergebnis:",cos(π/4)
⋮
```

Anzeigen von Informationen in einem Dialogfeld

Mit dem Befehl **Text** können Sie ein laufendes Programm unterbrechen und Informationen in einem Dialogfeld anzeigen. Der Benutzer kann dann **OK** wählen, um das Programm fortzusetzen, oder **Cancel**, um es abzubrechen.

In Funktionen ist der Befehl **Text nicht zulässig.**

```
⋮
Text "Fläche=" & area
⋮
```

Hinweis: Das Anzeigen eines Ergebnisses mit **Disp** oder **Text** bewirkt nicht, dass dieses auch gespeichert wird. Wenn Sie auf das Ergebnis später voraussichtlich noch einmal zugreifen müssen, legen Sie es in einer globalen Variablen ab.

```
⋮
cos(π/4)➞maximum
Disp maximum
⋮
```

Verwenden lokaler Variablen

Eine lokale Variable ist eine temporäre Variable, die nur so lange existiert, wie eine benutzerdefinierte Funktion ausgewertet oder ein benutzerdefiniertes Programm ausgeführt wird.

Beispiel für eine lokale Variable

Das folgende Programmsegment zeigt eine **For...EndFor-Schleife** (die weiter hinten in diesem Modul erläutert wird). Die Variable *i* ist der Schleifenzähler. In den meisten Fällen wird die Variable *i* nur genutzt, während das Programm ausgeführt wird.

```
Local i ❶
For i,0,5,1
  Disp i
EndFor
Disp i
```

❶ Legt die Variable *i* als lokale Variable fest.

Hinweis: Soweit möglich sollten Sie alle Variablen als lokale Variablen festlegen, die nur innerhalb des Programms genutzt werden und nach Beendigung des Programms nicht mehr benötigt werden.

Was verursacht eine Fehlermeldung "Nicht definierte Variable"?

Eine Fehlermeldung **Nicht definierte Variable** wird ausgegeben, wenn Sie eine benutzerdefinierte Funktion auswerten oder ein benutzerdefiniertes Programm ausführen, die/das sich auf eine lokale Variable bezieht, die nicht initialisiert wurde (der kein Wert zugewiesen wurde).

Beispiel:

```
Define fact(n)=Func
  Local m ❶
  While n>1
    n•m→m: n–1→n
  EndWhile
  Return m
  EndFunc
```

❶ Der lokalen Variable *m* wird kein Anfangswert zugewiesen.

Lokale Variablen initialisieren

Allen lokalen Variablen muss ein Anfangswert zugewiesen werden, bevor die referenziert werden können.

```
Define fact(n)=Func
  Local m: 1→m ❶
  While n>1
```

```
  n•m→m: n–1→n
EndWhile
Return m
EndFunc
```

❶ 1 wird als Anfangswert für *m* gespeichert.

Hinweis (CAS): Funktionen und Programme können keine lokale Variable verwenden, um symbolische Berechnungen durchzuführen.

CAS: Durchführen symbolischer Berechnungen

Wenn eine Funktion oder ein Programm symbolische Berechnungen durchführen soll, müssen Sie an Stelle einer lokalen Variable eine globale Variable verwenden. Sie müssen jedoch sicherstellen, dass die globale Variable nicht bereits außerhalb des Programms besteht. Dabei können die folgenden Methoden hilfreich sein.

- Geben Sie einen globalen Variablennamen (typischerweise mit zwei oder mehr Zeichen) an, dessen Vorhandensein außerhalb der Funktion / des Programms unwahrscheinlich ist.
- Fügen Sie **DelVar** in ein Programm ein, um die globale Variable, sofern sie bereits existiert, löschen, bevor Sie auf diese Bezug nehmen. (**DelVar** löscht keine geschützten oder verknüpften Variablen.)

Unterschiede zwischen Funktionen und Programmen

Im Programmeditor definierte Funktionen ähneln den Funktionen, die in der TI-Nspire™ Software enthalten sind.

- Funktionen müssen ein Ergebnis ausgeben, das sich grafisch darstellen oder in eine Tabelle eingeben lässt. Programme können kein Ergebnis ausgeben.
- Eine Funktion kann innerhalb eines Ausdrucks verwendet werden (ein Programm hingegen nicht). Beispiel: **3 • func1(3)** ist gültig, **3 • prog1(3)** hingegen nicht.
- Die Ausführung von Programmen ist nur in den Applikationen Calculator und Notes möglich. Funktionen können in Calculator, Notes, Lists & Spreadsheet, Graphs & Geometry und Data & Statistics ausgewertet werden.
- Eine Funktion kann sich auf jede Variable beziehen; sie kann einen Wert jedoch nur in einer lokalen Variable speichern. Programme können in lokalen und globalen Variablen speichern.

  Hinweis: Argumente, die zur Übergabe von Werten an eine Funktion verwendet werden, werden automatisch als lokale Variablen behandelt. Wenn Sie sie in anderen Variablen speichern möchten, müssen Sie sie innerhalb der Funktion als **Lokal(Local)** festlegen.
- Eine Funktion kann kein Programm als Subroutine aufrufen, jedoch kann es eine andere benutzerdefinierte Funktion aufrufen.
- Sie können kein Programm innerhalb einer Funktion definieren.

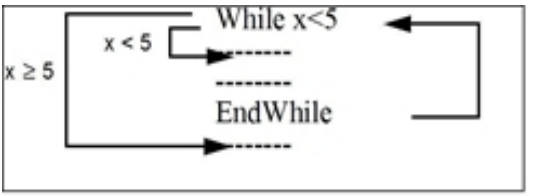
- Eine Funktion kann keine globale Funktion definieren, sie kann jedoch eine lokale Funktion definieren.

Aufrufen eines Programms aus einem anderen Programm

Ein Programm kann ein anderes Programm als Subroutine aufrufen. Eine Subroutine kann als externe (separates Programm) oder als interne Subroutine (im Hauptprogramm enthalten) angelegt sein. Subroutinen sind nützlich, wenn ein Programm dieselbe Gruppe von Befehlen an verschiedenen Stellen wiederholen muss.

Aufrufen eines separaten Programms

Zum Aufrufen eines separaten Programms verwenden Sie dieselbe Syntax wie zum Ausführen des Programms aus der Eingabezeile.



Definieren und Aufrufen einer internen Subroutine

Zum Definieren und Aufrufen einer internen Subroutine verwenden Sie den Befehl **Definiere (Define)** zusammen mit **Prgm...EndPrgm**. Da eine Subroutine definiert sein muss, bevor sie aufgerufen werden kann, sollten Subroutinen bereits am Anfang des Hauptprogramms definiert werden.

Eine interne Subroutine wird genauso aufgerufen und ausgeführt wie ein separates Programm.

```
Define subtest1()=
  Prgm
  local subtest2 ❶
  Define subtest2(x,y)= ❷
  Prgm
    Disp x,y
  EndPrgm
  ©Anfang des Hauptprogramms
  For i,1,4,1
    subtest2(i,I*1000) ❸
  EndFor
  EndPrgm
```

❶ Legt die Subroutine zu einer lokalen Variable fest.

❷ Definiert die Subroutine.

❸ Ruft die Subroutine auf.

Hinweis: Verwenden Sie das Programmeditor-Menü **Var**, um die Befehle **Definiere (Define)** und **Prgm...EndPrgm** einzugeben.

Hinweise zur Verwendung von Subroutinen

Am Ende der Subroutine wird das aufrufende Programm fortgesetzt. Um eine Subroutine an einem anderen Zeitpunkt zu beenden, verwenden Sie **zurück (Return)** ohne Argument.

Eine Subroutine kann nicht auf lokale Variablen zugreifen, die im aufrufenden Programm festgelegt wurden. Ebenso kann das aufrufende Programm nicht auf lokale Variablen zugreifen, die in einer Subroutine festgelegt wurden.

Lbl Befehle sind lokale Befehle für die Programme, in denen sie sich befinden. Daher kann ein **Gehe zu (Goto)**-Befehl im aufrufenden Programm nicht zu einer Marke in einer Subroutine springen oder andersherum.

Vermeidung zirkulärer Definitionsfehler

Bei der Auswertung einer benutzerdefinierten Funktion oder der Ausführung eines Programms können Sie ein Argument angeben, das dieselbe Variable enthält, die für die Definition der Funktion bzw. für die Erstellung des Programms verwendet wurde. Um jedoch zirkuläre Definitionsfehler zu vermeiden, müssen Sie Variablen, die bei der Auswertung der Funktion oder der Ausführung des Programms verwendet werden, einen Wert zuweisen. Beispiel:

```
x+1→x ❶
```

– oder –

```
For i,i,10,1
  Disp i ❶
EndFor
```

❶ Verursacht eine Fehlermeldung **Zirkuläre Definition (Circular definition)**, wenn für x oder i kein Wert vorliegt. Der Fehler tritt nicht auf, wenn x bzw. i bereits ein Wert zugewiesen wurde.

Steuerung des Ablaufs einer Funktion / eines Programms

Wenn Sie ein Programm ausführen oder eine Funktion auswerten, werden die Programmzeilen der Reihe nach abgearbeitet. Es gibt jedoch einige Befehle, mit denen der Programmablauf verändert werden kann. Beispiel:

- Steuerungsstrukturen wie z.B. **If...EndIf**-Befehle beinhalten einen bedingten Test, um zu entscheiden, welcher Teil eines Programms ausgeführt werden soll.
- Schleifenbefehle wie **For...EndFor** wiederholen eine Gruppe von Befehlen.

Verwenden von If, Lbl und Goto zur Steuerung des Programmablaufs

Mit dem Befehl **If** und einigen **If...EndIf**-Strukturen können Sie die Ausführung einer Anweisung oder eines Blocks von Anweisungen an Bedingungen knüpfen, d.h. die Ausführung hängt vom Ergebnis eines Tests ab (z.B. x>5). Die Befehle **Lbl** (Marke) und **Gehe zu (Goto)** ermöglichen es, von einer Stelle des Programms zu einer anderen Stelle zu verzweigen oder zu springen.

Den Befehl **If** und mehrere **If...EndIf**-Strukturen finden Sie im Programmeditor-Menü **Steuerung (Control)**.

Wenn Sie eine Struktur wie z.B. **If...Then...EndIf** einfügen, wird an der Cursorposition eine Vorlage eingefügt. Der Cursor wird so positioniert, dass Sie einen bedingten Test eingeben können.

If-Befehl

Um nach einem wahren bedingten Test einen einzelnen Befehl auszuführen, verwenden Sie die allgemeine Form:

```
If x>5
  Disp "x ist größer als 5"  ❶
Disp x  ❷
```

❶ Wird nur ausgeführt, wenn x>5; anderenfalls wird der Punkt übersprungen.

❷ Zeigt immer den Wert von x an.

In diesem Beispiel müssen Sie für x einen Wert speichern, bevor Sie den Befehl **If** ausführen.

If...Then...EndIf-Strukturen

Um eine Gruppe von Befehlen auszuführen, wenn ein bedingter Test wahr ist, verwenden Sie die Struktur:

```
If x>5 Then
  Disp "x ist größer als 5"  ❶
  2•x→x  ❶
EndIf
Disp x  ❷
```

❶ Wird nur ausgeführt, wenn x>5.

❷ Zeigt folgende Werte an:
- 2x, wenn x>5
- x, wenn x≤5

Hinweis: EndIf markiert das Ende des **Then** Blocks, der ausgeführt wird, wenn die Bedingung wahr ist.

If...Then...Else... EndIf-Strukturen

Um eine Gruppe von Befehlen auszuführen, wenn ein bedingter Test wahr ist und eine andere Gruppe auszuführen, wenn die Bedingung falsch ist, verwenden Sie die folgende Struktur:

```
If x>5 Then
  Disp "x ist größer als 5"  ❶
  2•x→x  ❶
Else
  Disp "x ist kleiner als oder gleich 5"  ❷
  5•x→x  ❷
EndIf
Disp x  ❸
```

❶ Wird nur ausgeführt, wenn x>5.

❷ Wird nur ausgeführt, wenn x≤5.

❸ Zeigt folgende Werte an:
2x, wenn x>5
5x, wenn x≤5

If...Then...ElseIf... EndIf-Strukturen

Eine komplexere Form des If-Befehls kann mehrere Bedingungen überprüfen. Angenommen, Sie möchten ein Programm erstellen, das ein vom Benutzer eingegebenes Argument auf vier Optionen prüfen soll.

Um das Argument auf jede Option (If Möglichkeit=1, If Möglichkeit=2, usw.) zu überprüfen, verwenden Sie die Struktur **If...Then...ElseIf...EndIf**.

Lbl und Goto Befehle

Sie können den Ablauf auch über die Befehle **Lbl** (Marke) und **Goto** steuern. Diese Befehle finden Sie im Programmeditor-Menü **Übertragungen (Transfers)**.

Verwenden Sie den Befehl **Lbl**, um eine bestimmte Stelle in der Funktion oder im Programm zu markieren (d.h. ihr einen Namen zuzuweisen).

Lbl *MarkeName*	Name, der dieser Stelle zugewiesen wird (Verwenden Sie die gleiche Namenskonvention wie beim Variablen Namen)

Anschließend können Sie den Befehl **Goto** an jeder Stelle der Funktion / des Programms verwenden, um zu der Stelle zu verzweigen, die der genannten Marke entspricht.

Goto *MarkeName*	gibt an, zu welchem **Lbl** Befehl verzweigt werden soll

Da ein **Goto** Befehl an keine Bedingungen geknüpft ist (verzweigt immer zur angegebenen Marke), wird er oft gemeinsam mit einem **If**-Befehl verwendet, sodass Sie einen bedingten Test angeben können. Beispiel:

```
If x>5
  Goto GT5  ❶
Disp x
--------
--------  ❷
Lbl GT5
Disp "Die Zahl war > 5"
```

❶ Wenn x>5, wird direkt zu Marke GT5 verzweigt.

❷ In diesem Beispiel muss das Programm Befehle enthalten (wie z.B. **Stop**), die verhindern, dass **Lbl** GT5 ausgeführt wird, wenn x≤5.

Verwenden von Schleifen zum Wiederholen einer Gruppe von Befehlen

Um dieselbe Gruppe von Befehlen nacheinander zu wiederholen, verwenden Sie eine der Schleifenstrukturen. Es stehen mehrere Schleifentypen zur Verfügung. Jeder Typ bietet eine andere Möglichkeit, die Schleife über einen bedingten Test zu verlassen.

Schleifen und schleifenbezogene Befehle finden Sie in den Programmeditor-Menüs **Steuerung (Control)** und **Übertragungen (Transfers)**.

Wenn Sie eine der Schleifen-Strukturen einfügen, wird die entsprechende Vorlage an der Cursorposition eingefügt. Sie können dann damit beginnen, die Befehle einzugeben, die innerhalb der Schleife ausgeführt werden.

For...EndFor-Schleifen

Eine **For...EndFor**-Schleife verwendet einen Zähler, um die Anzahl der Schleifenwiederholungen zu kontrollieren. Die Syntax des Befehls **For** lautet:

Hinweis: Der Endwert kann kleiner sein als der Anfangswert, wenn die Erhöhung negativ ist.

For *Variable*, *Anfang*, *Ende* [, *Erhöhung*]

❶ ❷ ❸ ❹

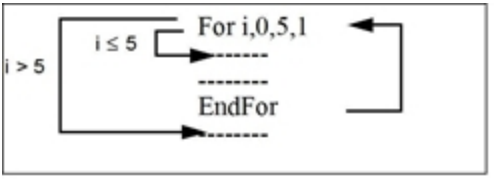
❶ *Variable* als Zähler

❷ Zählerwert bei der ersten Ausführung von **For**

❸ Verlässt die Schleife, wenn *Variable* diesen Wert überschreitet

❹ Wird dem Zähler bei jeder Wiederholung von **For** hinzugefügt (Wenn dieser optionale Wert ausgelassen wird, ist die *Erhöhung* 1.)

Wenn **For** ausgeführt wird, wird der Wert *Variable* mit dem Wert *Ende* verglichen. Wenn hierbei *Variable* nicht höher ist als *Ende*, wird die Schleife ausgeführt; anderenfalls springt die Steuerung zu dem Befehl, der auf **EndFor** folgt.



Hinweis: Der Befehl **For** erhöht den Zähler Variable automatisch, sodass die Funktion / das Programm die Schleife nach einer bestimmten Anzahl an Wiederholungen verlassen kann.

Am Ende der Schleife (**EndFor**) springt die Steuerung zurück zum Befehl **For**, wo der Wert Variable erhöht und mit *Ende* vergleichen wird.

Beispiel:

```
For i,0,5,1
  Disp i ❶
EndFor
Disp i ❷
```

❶ Zeigt 0, 1, 2, 3, 4 und 5 an.

❷ Zeigt 6 an. Wenn *Variable* auf 6 erhöht wird, wird die Schleife nicht mehr ausgeführt.

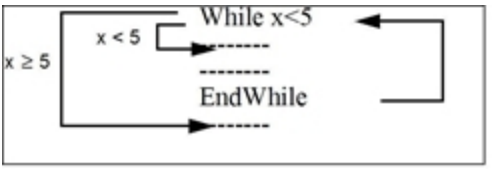
Hinweis: Wenn die Zählervariable nach Beendigung der Funktion / des Programms nicht gespeichert werden muss, können Sie die Variable als lokal festlegen.

While...EndWhile-Schleifen

Eine **While...EndWhile**-Schleife wiederholt einen Block von Befehlen so lange, wie eine festgelegte Bedingung wahr ist. Die Syntax des Befehls **While** lautet:

While *Bedingung*

Während der Ausführung von **While** wird die *Bedingung* ausgewertet. Wenn die *Bedingung* wahr ist, wird die Schleife ausgeführt; anderenfalls springt die Steuerung zu dem Befehl, der auf **EndWhile** folgt.



Hinweis: Der Befehl **While** ändert nicht automatisch die Bedingung. Sie müssen Befehle einfügen, die es der Funktion / dem Programm ermöglichen, die Schleife zu verlassen.

Am Ende der Schleife (**EndWhile**) springt die Steuerung zurück zum Befehl **While**, wo die Bedingung erneut ausgewertet wird.

Um die Schleife beim ersten Mal ausführen zu können, muss die Bedingung am Anfang wahr sein.

- Alle Variablen, auf die in der Bedingung Bezug genommen wird, müssen vor dem Befehl **While** eingegeben werden. (Sie können die Werte in die Funktion bzw. das Programm einbauen oder den Benutzer zur Eingabe der Werte auffordern.)
- Die Schleife muss Befehle enthalten, die die Werte in der Bedingung ändern und so die Bedingung letztendlich unwahr machen. Anderenfalls wäre die Bedingung immer wahr und die Funktion / das Programm könnte die Schleife nicht verlassen (Endlosschleife).

Beispiel:

```
0→x ❶
While x<5
  Disp x ❷
  x+1→x ❸
EndWhile
Disp x ❹
```

❶ Stellt x anfänglich ein.

❷ Zeigt 0, 1, 2, 3 und 4 an.

❸ Erhöht x.

❹ Zeigt 5 an. Wenn x auf 5 angewachsen ist, wird die Schleife nicht mehr ausgeführt.

Loop...EndLoop-Schleifen

Loop...EndLoop erstellt eine Endlosschleife, die unendlich wiederholt wird. Der Befehl **Loop** enthält keine Argumente.

Typischerweise fügen Sie Befehle in die Schleife ein, die es dem Programm ermöglichen, die Schleife zu verlassen. Häufig verwendete Befehle hierfür sind: **If**, **Exit**, **Goto** und **Lbl** (Marke). Beispiel:

```
0→x
Loop
  Disp x
  x+1→x
  If x>5 ❶
    Exit
EndLoop
Disp x ❷
```

❶ Ein **If**-Befehl überprüft die Bedingung.

❷ Verlässt die Schleife und springt hierher, wenn x auf 6 anwächst.

Hinweis: Mit dem Befehl **Exit** wird die aktuelle Schleife verlassen.

In diesem Beispiel kann sich der **If**-Befehl an einer beliebigen Stelle in der Schleife befinden.

Wenn der If-Befehl:	wird die Schleife:
am Anfang der Schleife ist	nur ausgeführt, wenn die Bedingung wahr ist.
am Ende der Schleife ist	mindestens ein Mal ausgeführt und nur wiederholt, wenn die Bedingung wahr ist.

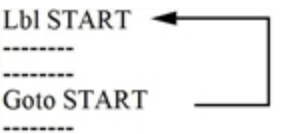
Der Befehl **If** kann auch den Befehl **Goto** verwenden, um die Programmsteuerung zu einem bestimmten **Lbl** (Marke)-Befehl zu übertragen.

Sofortiges Wiederholen einer Schleife

Der Befehl **Cycle** überträgt die Programmsteuerung sofort an die nächste Wiederholung der Schleife (bevor die aktuelle Wiederholung beendet ist). Dieser Befehl funktioniert bei **For...EndFor**, **While...EndWhile** und **Loop...EndLoop**.

Lbl- und Goto-Schleifen

Obwohl die Befehle **Lbl** (Marke) und **Goto** keine ausschließlichen Schleifenbefehle sind, können sie benutzt werden, um eine Endlosschleife zu erzeugen. Beispiel:



Wie bei **Loop...EndLoop** muss die Schleife Befehle enthalten, die es der Funktion / dem Programm ermöglichen, die Schleife zu verlassen.

Ändern der Moduseinstellungen

Funktionen und Programme können die Funktion **setMode()** verwenden, um vorübergehend bestimmte Kalkulations- oder Ergebnis-Modi einzustellen. Das Programmeditor-Menü **Modus (Mode)** erleichtert die Eingabe der korrekten Syntax, ohne dass Sie hierfür numerische Codes benötigen.

Hinweis: Modusänderungen, die in einer Funktions- oder Programmdefinition vorgenommen werden, haben außerhalb der Funktion / des Programms keinen Bestand.

Einstellen eines Modus

1. Platzieren Sie den Cursor an der Stelle, an der Sie die Funktion **setMode** einfügen möchten.
2. Wählen Sie im Menü **Modus** den zu ändernden Modus und wählen Sie die neue Einstellung aus.

   Die korrekte Syntax wird an der Position des Cursors eingefügt. Beispiel:

```
setMode(1,3)
```

Behebung von Programm- und Bedienungsfehlern

Nachdem Sie eine Funktion / ein Programm erstellt haben, können Sie mit mehreren Techniken nach Fehlern suchen und diese korrigieren. Sie können außerdem einen Befehl zur Fehlerbehandlung in die Funktion / das Programm selbst einbauen.

Wenn Ihre Funktion / Ihr Programm dem Benutzer eine Wahl zwischen mehreren Optionen gestattet, stellen Sie sicher, dass Sie jede Option getestet haben.

Techniken für die Fehlerbehebung

Meldungen zu Laufzeitfehlern können Syntaxfehler lokalisieren, jedoch keine Fehler in der Programmlogik. Hierbei können folgende Techniken hilfreich sein:

- Fügen Sie vorübergehend **Disp**-Befehle ein, um die Werte kritischer Variablen anzuzeigen.
- Um zu überprüfen, dass eine Schleife so oft wiederholt wird wie gewünscht, verwenden Sie **Disp**, um die Zählervariable oder die Werte des bedingten Tests anzuzeigen.
- Um zu überprüfen, ob eine Subroutine ausgeführt wird, verwenden Sie **Disp**, um Meldungen wie “Start der Subroutine (Entering subroutine)” und “Ende Subroutine (Exiting subroutine)” am Anfang und Ende der Subroutine anzuzeigen.
- So brechen Sie ein Programm oder eine Funktion manuell ab:
  - Windows®: Halten Sie die Taste **F12** gedrückt und drücken Sie mehrmals **Enter**.
  - Macintosh®: Halten Sie die Taste **F5** gedrückt und drücken Sie mehrmals **Enter**.
  - Handheld: Halten Sie die Taste [on] gedrückt und drücken Sie mehrmals [enter].

Befehle zur Fehlerbehandlung

Befehl	Beschreibung
Try...EndTry	Definiert einen Block, das eine Funktion / ein Programm einen Befehl ausführen lässt und eine Wiederherstellung ermöglicht, falls der Befehl einen Fehler erzeugt.
ClrErr	Löscht den Fehlerstatus und setzt die Systemvariable *errCode auf Null. Ein Beispiel für die Verwendung von* errCode *finden Sie unter dem Befehl* ***Try im Referenzhandbuch****.*
PassErr	Übergibt einen Fehler an die nächste Stufe des **Try...EndTry**-Blocks.

Allgemeine Informationen

Online-Hilfe

education.ti.com/eguide

Wählen Sie Ihr Land aus, um weitere Produktinformationen zu erhalten.

Kontakt mit TI Support aufnehmen

education.ti.com/ti-cares

Wählen Sie Ihr Land aus, um auf technische und sonstige Support-Ressourcen zuzugreifen.

Service- und Garantieinformationen

education.ti.com/warranty

Wählen Sie Ihr Land aus, informationen zur Dauer und zu den Bedingungen der Garantie bzw. zum Produktservice zu erhalten.

Eingeschränkte Garantie. Diese Garantie hat keine Auswirkungen auf Ihre gesetzlichen Rechte.