Einführung in das Progammieren mit TI Nspire

## Einführung+Verzweigung

**Einführung in das Programmieren, 1. Verzweigungen** Haftendorn 2011

Stückweise definierte Funktionen erstellt man am einfachsten mit der Vorlage

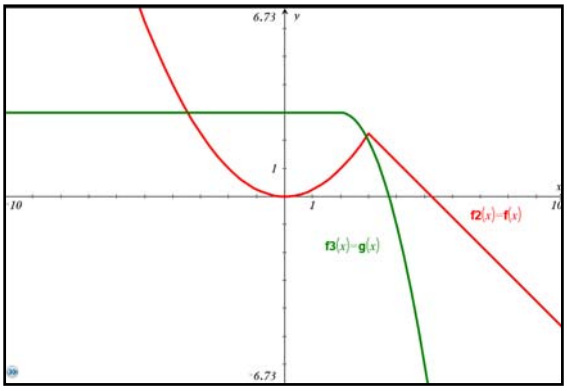
$g(x):=\begin{cases}3, & x<2\\ -(x-2)^2+3, & x\geq 2\end{cases}$ • *Fertig* Diese Vorlage gibt es auch mit drei oder mehr Zeilen.

Dieselbe Wirkung erreicht man durch eigene Programmierung. Kann man in eine Mathezelle Vorschläge kopieren und variieren oder Im Katalog bei Funktionen Programme sich Anregungen holen. (Neue Zeile mit Alt+Enter)

```
Define f(x)=Func                    • Fertig   SieheFunktionenim Graphfenster
  If x<3 Then
    Return x^2/4
  Else
    Return -(x-3)+9/4
  EndIf
EndFunc
```

Achtung: Die Zeilen müssen so sein. Enger kann man nicht schreiben.

1.1




1.2

Wenn man nun aber nicht ganz gewöhnliche Funktionen haben will, ist eigenes Programmieren die einzige Möglichkeit.

```
Define fz(x)=Func                   • Fertig   fz(3) • Ziffer   fz(12) • Zahl
  If 0≤x and x<10 Then
    Return "Ziffer"
  Else
    Return "Zahl"
  EndIf
EndFunc
```

In den Blöcken zwischen Then und Else bzw. Else und EndIf können beliebig viele Befehle stehen. Schreibt man mehrere in eine Zeile, trennt man die Befehle durch Doppelpunkte. (Neue Zeile im Programm Alt+Enter, falls im Notes-Fenster)

```
Define eig(n)=Func                  • Fertig
  Disp "n=",n," Quadrat=",n^2," Kubus=",n^3
  Disp "Rechne selbst: ": Return randInt(1,20)
EndFunc
```

Ausgaben mit Disp (display) kann man nur im Calculatorfenster sehen.

1.3

```
eig(11)
                          n=11 Quadrat=121 Kubus=1331
                          Rechne selbst:
                                                    5
eig(5)
                          n=5 Quadrat=25 Kubus=125
                          Rechne selbst:
                                                   10
eig(6)
                          n=6 Quadrat=36 Kubus=216
                          Rechne selbst:
                                                    2
                                                 3/99
```

1.4

Für Funktionen eignet sich noch Notes, für Programme braucht man den **Programmeditor**.

Da ist das Zahlenraten geschrieben. Hier ist es eine Extraseite.

Siehe diese an, mache ein neues Calculatorfenster auf und schreibe

dort rate() gefolgt von Enter. Spiele damit!

**rate**() • Fehler: Variable ist nicht definiert Das passiert, weil im Notesfenster das Eingabefenster, das mit dem Befehl request erzeugt wird, nicht aufgeht.

Den Progammeditor bekommt man aus dem Menu durch Einfügen->Programmeditor.

Nun muss man sofort einen Namen für das Programm, bzw. die Funktion, eingeben.

Hier also rate, ohne die Klammern. Dann muss man noch entscheiden, ob man ein Programm oder eine Funktion will.

Der Unterschied ist: **Funktionen** haben einen "Rückgabewert", den man mit dem Befehl return am Ende ausgibt. **Programme** können allerlei tun, Werte anfordern, verarbeiten, anzeigen mit Disp usw. aber mit der Ausgabe eines Programms kann man nichts machen. Mit Funktionsausgaben kann man rechnen, wenn es entsprechende Objekte sind. Sie funktionieren im Notesfenster, Programme oft nicht.

Im Programmeditor hat man den Vorteil, dass in der Werkzeugpalette alle Programmierelemente übersichtlich zum Übernehmen zur Verfügung stehen. Für Anfänger ist das eine große Hilfe.

1.5

```
* rate                                                    4/16
Define rate()=
Prgm
Local a,dein,ant
  a:=randInt(1,20)
  Lbl start
  Request "Gib eine Zahl bis 20 ein:  ",dein
  If dein<a Then
    Disp "zu klein "
  ElseIf dein>a Then
      Disp "zu groß "
  Else
    Disp "Super, getroffen "
  EndIf
  RequestStr "Weitermachen j/n ",ant
  If ant="j" Then
    Goto start
  EndIf
  Disp "Danke"
EndPrgm
```

1.6

```
rate()
 Gib eine Zahl bis 20 ein:  10
 zu klein
 Weitermachen j/n j
 Gib eine Zahl bis 20 ein:  15
 zu klein
 Weitermachen j/n j
 Gib eine Zahl bis 20 ein:  17
 Super, getroffen
 Weitermachen j/n n
 Danke
                                                   Fertig
© Dies ist ein Calculator-Fenster •
                                                    2/99
```

1.7

**Variablengebrauch** in Funktionen und Programmen

Bei der Definition von f, fz, eig, rate … schreibt man in die erscheinenden Klammern die "Argumente", also Variabennamenfür das Urbild der Funktion bzw. die Eingabefür das Programm. Man erhält f(x), fz(x) ,eig(n), rate(). Erhätten auch mehrere Argumente sein können: v(la,br,h)… Rate() hat kein Argument, die Klammern müssen aber da stehen. Ob die Variablen für Zahlen, Text, Matrizen, Listen stehen, wird in der Programmierung geklärt und muss dem Nutzer durch den Kontext klar sein.

**Globale Variable** sind alle Variablen eines "Problems"

**Lokale Variable** werden nach der Define-Zeile hinter dem Wort **local** geschrieben.

Bei rate() ist das z.B. a. Hier wird in der nächsten Zeile dem a ein Wert zugewiesen, der ein im Problem etwa schon vorhandenes a gar nicht stört.

Guter Stil ist, alle bei der Programierung verwendeten Variablen als local zu vereinbaren.

Wenn globale Variable im Programm gebraucht werden, sollten sie besser als Argumente beim Aufruf übergeben werden.

1.8

Einführung in das Progammieren mit TI Nspire

## Schleifen

**Programmierelement: Schleifen** Haftendorn 2011

In allen Programmiersprachen gibt Zählschleifen, bedingte und unbedingte Schleifen und Sprünge. Hier findet man sie bei Mathematische Operatoren-->Funktionen und Programme-->Steuerung. Beim Anklicken sieht man darunter die Verwendung. Hier folgen Beispiele im Text und echt in den Programmen **Schleifen**(n) und **sprung()**

**For** i, 1, 20 Der Anweisungsblock wird nun 20 mal durchlaufen.
Block Zählschleife, wenn man vorher weiß, wie oft man es braucht.
**EndFor**

**While** z > 0.01 Das z muss vorher schon einen Wert haben.
Block, der ganz sicher das z immer kleiner macht
**EndWhile** Vorsicht, meist sind =-Bedingungengefährlich

**Loop**
Block mit sicherem Ausstieg durch Exit oder einen anderen Sprung
**EndLoop**

**Lbl** meineMarke irgedwo steht **GoTo** meineMarke

2.1

**Was ist bei Fehlern zu tun?**

Wenn man die nötigen Austiege nicht richtig gemacht hat, hilft:

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 c gedrückt und drücken Sie mehrmals Enter-Taste.

Es werden noch weitere Möglichkeiten eröffnet, mit denen man "**Fehler abfangen**" kann.

Try Block Else Block EndTry (in 5 Zeilen

Zu ClrErr PassErr errCode warnCodes(....) Liest man im Referenzhandbuch, am besten unter dem Stichwort endtry

2.2

schleifen 8/13

```
Define LibPub schleifen(n)=
Prgm
 Local i,li,z
  li:={}
© Zählschleife
  For i,1,n
    li:=augment(li,{i^2})
  EndFor
  Disp li
© While-Schleife, Bedingung im Kopf
 i:=1:li:={}
 While i≤n
   li:=augment(li,{i^3}):   i:=i+1
 EndWhile
 Disp li
EndPrgm
```

2.3

sprung 11/15

```
Define sprung()=
Prgm
Local i,li,z
© Loop-Schleife, Bedingung innen oder hinten
  li:={}
 Loop
  z:=randInt(1,9):   li:=augment(li,{z})
  If z=6 Then
    Disp li,"danke, das war die erste gewürfelte 6 ":   Exit
  EndIf
 EndLoop
 Lbl nochmal
 z:=randInt(1,9):   li:=augment(li,{z})
 If z>1 Then
   Goto nochmal
 EndIf
 Disp li,"danke, das war die erste 1 nach der 6 "
EndPrgm
```

2.4

```
schleifen(10)
                    {1,4,9,16,25,36,49,64,81,100}
                    {1,8,27,64,125,216,343,512,729,1000}
                                                  Fertig
sprung()
{1,1,7,7,8,4,2,5,9,7,5,6} danke, das war die erste gewürfelte 6
{1,1,7,7,8,4,2,5,9,7,5,6,9,7,8,5,4,2,6,8,2,5,5,3,8,9,5,4,3,6,6,8,7,5,9,6,4,5,8,8,3,7,5,5,2,} danke, das war die erste 1 nach der 6
                                                  Fertig
sprung()
                    {9,8,3,7,3,1,7,2,6} danke, das war die erste gewürfelte 6
                    {9,8,3,7,3,1,7,2,6,8,9,5,1} danke, das war die erste 1 nach der 6
                                                  Fertig
```

3/99

2.5

**Nochmal etwas zu den Variablen ..**

Belegen wir i global i:=10 • 10 . **Schnelle Listenerzeugung**

seq(i^2,i,1,12) • {1,4,9,16,25,36,49,64,81,100,121,144} und nun **i** • 10

seq ist eine Funktion mit 4 Argumenten, ein Berechnungsterm, eine Laufvariable, hier auch i, start und ende. Wie bei allen Funktionen sind die Argumente nach dem Aufruf lokal, sie werden außen nicht geändert. Darum ist das vorher belegte i immernoch 10. Kleine Unersuchung: übrigens Func aus Math. Operatoren holen:

```
Define test(n)=Func          • Fertig    test(5) • [10  6] und i • 10
                Local i,a
                For i,1,n
                  a:=2·i
                EndFor
                Return [a  i]
              EndFunc
```

Ohne die local-Zeile ging es nicht!!!!! Zum Schluss ist innen a=10 und i=6, außen aber gilt immernoch i=10. Man sieht auch, dass die Schleife mit i=5 zuletzt durchlaufen wurde, dann wurde i=6 und bei For geklärt, dass es zuende ist.

2.6

## Eingabe+Ausgabe, Steuerung

**Steuerung und Eingriffe von "außen"**

findet man beim Unterpunkt "Übertragung" und bei E/A (Eingabe/Ausgabe)

**Return z** Wird in Funktionen verwendet, im 1. Problem bei f und fz.

Nach der Ausführung von Return wird die Funktion (oder die Unterfunktion) verlassen.

**Cycle** verkürzt Schleifen und spring sofort zum Schleifenanfang und man mit einem neuen Durchlauf weiter.

**Exit** führt zum sofortigen Ende der Schleife.

**Stop** hält das Programm an.

**Lbl** und **GoTo** sind bei den Schleifen und Sprüngen erklärt.

**Disp** heißt display und sorgt komfortabel für eine Anzeige aber nur im Calculator-Fenster, nicht im Notes-Fenster. Ist oben schon oft werwendet.

Abfrage heißt **Request** nimmt Zahlen entgegen, **RequestStr** für Zeichenketten,

ist oben bei **rate()** verwendet. Das im Fenster Eingegebenewird in der Variablen gespeichert, die man nach dem Komma nennt. Dieses klappt auch nicht im Notesfenster.

3.1

Die Möglichkeiten von Request sind erst mit der Version 3 eingebaut worden. Der TI voyage und sogar seine Vorläufer hatten die Möglichkeit, der TI Nspire zunächst nicht. Es ist gut dass es da jetzt gibt.

Die Variablenbehandlung war bei den Modellen vor dem TI Nspire außerordentlich unfreundlich. Das ist jetzt supergut gelöst:

Also: In einem Dokument kann man mehrere "Probleme" eröffnen. Jedes hat einen Variablenschutz gegenüber den anderen Problemen des Dokument es. Erst recht stören sich verschiedene geöffnete Dokumente nicht.

rate() • rate() Man sieht, das Programm rate() aus dem 1. Problem gibt es hier nicht.

In einem Problem kann man mit local Variablen von Funktionen und Programmen nach außen abschotten.

Wenn man aber selbst definierte Funktionen umfassend in mehrern Dokumenten verwenden will, greift man auf das Konzept der "**Bibliotheken**" zurück.

Das ist auf dieser Site bei Kryptografie ausführlich getan und dort auch erklärt.

Lesen Sie bei Bedarf im Referenzhandbuch das Kapitel: Bibliotheken.

3.2

Einführung in das Progammieren mit TI Nspire

## Onkel Otto und Listen

**Listenoperationen**

eingebaute Listenoperationen

Beispielliste li:={a,bra,ka,da,bra} ▸ {a,bra,ka,da,bra}

dim(liste) gibt die Länge der Liste, die Anzahl der Elemente an.

dim(li) ▸ 5 left(li,2) ▸ {a,bra} right(li,2) ▸ {da,bra}

left(liste,k) gibt die linken k Elemente als Liste zurück

right(liste,k) gibt die letzten k Elemente als Liste zurück

augment(liste1,liste2) fügt die beiden Listen zusammen und gibt sie als **eine** Liste aus.

lis:=augment(li,{40,räuber}) ▸ {a,bra,ka,da,bra,40,räuber}

mid(liste, k, r) gibt ab Platz k r Elemente aus.

mid(liste,k,1) hätte wohl auch das k-te Element geliefert

mid(lis,3,2) ▸ {ka,da} mid(lis,6,1) ▸ {40}

4.1

```
Define LibPub element(liste,k)=Func                    ▸ Fertig
  Local n
  n:=dim(liste)
  If k>n or n=0 Then
    Disp "geht nicht: ",n,"Elemente"
  Else
    Return right(left(liste,k),1)
  EndIf
EndFunc
```

Diese Funktion gibt das k-te-Element aus der Liste heraus.

element(lis,4) ▸ {da} lis ▸ {a,bra,ka,da,bra,40,räuber}

Die nächste Funktion löscht das k-te Element. **el**ement **del**ete

eldel(lis,7) ▸ {a,bra,ka,da,bra,40} lis ▸ {a,bra,ka,da,bra,40,räuber}

```
Define LibPub eldel(liste,k)=Func                      ▸ Fertig
  Return augment(left(liste,k-1),right(liste,dim(liste)-k))
EndFunc
```

4.2

Diese Funktion permutiert eine Liste einmal zufällig

```
Define LibPub perm(liste)=Func                         ▸ Fertig
  Local i,e,z,n,pli
  n:=dim(liste): pli:={}
  For i,1,n-1
    z:=randInt(1,dim(liste))
    e:=element(liste,z)
    pli:=augment(pli,e)
    liste:=eldel(liste,z)
  EndFor
  Return augment(pli,liste)
EndFunc
```

perm({1,2,3,4,5}) ▸ {3,2,5,4,1}

perm({1,2,3,4,5}) ▸ {4,1,3,5,2}

4.3

```
otto:={"onkel","otto","saß","weinend","in der Badewanne"}
 ▸ {"onkel","otto","saß","weinend","in der Badewanne"}
perm(otto) ▸ {"in der Badewanne","onkel","weinend","saß","otto"}
perm(otto) ▸ {"otto","in der Badewanne","onkel","weinend","saß"}
perm(otto) ▸ {"in der Badewanne","onkel","otto","saß","weinend"}
perm(otto) ▸ {"weinend","onkel","otto","saß","in der Badewanne"}
perm(otto) ▸ {"otto","in der Badewanne","saß","weinend","onkel"}
perm(otto) ▸ {"weinend","otto","onkel","saß","in der Badewanne"}
perm(otto) ▸ {"otto","saß","weinend","in der Badewanne","onkel"}
perm(otto) ▸ {"saß","weinend","onkel","in der Badewanne","otto"}
perm(otto) ▸ {"otto","saß","weinend","in der Badewanne","onkel"}
perm(otto) ▸ {"weinend","saß","in der Badewanne","onkel","otto"}
5! ▸ 120
```

4.4

**Anmerkungenzu LibPub und LibPiv in den Definitionen ..**

Der Einschub LibPub hat hier gar keine Wirkung. Er entsprang meiner Vorstellung, dass diese beiden Funktionen von mir auch in anderen Dokumenten nützlich wären.

Dazu muss aber folgendes passieren:

1. Ich müsste ein Dokument mit einem passenden Namen, z.B. tool.tns

im Verzeichnis mylib speichern.

2. Im ersten Problem dieses Dokumentes müsste ich diese beiden Definitionen unterbringen und nochmal mit re-Maus Syntax und Speichern sichern und das ganze Dokument nochmal speichern.

3. Die "Bibliotheken aktualisieren" (Bei Extras, bzz mit Crl Menu)

4. In jedem beliebigen Dokument mit "Langnamen"aufrufen also z.B. tool\perm(meineListe) Den Backslash auf dem Handheld mit crtl ÷ ,

anstelle von 4: Registerkarte Bibliotheken, tool auswählen, perm auswählen.

Übrigens: in der Definition kann man als zweite Zeile hinter dem Sonderzeichen

© Hilfen "Liste eingeben".© kann im Calculator für Kommentare stehen.

4.5