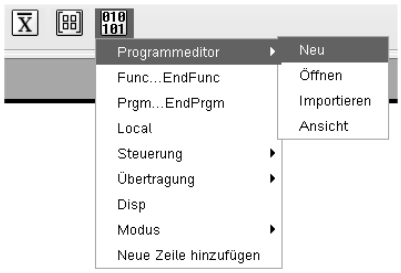
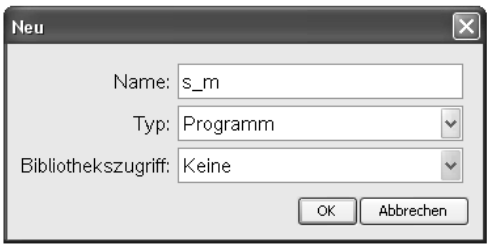
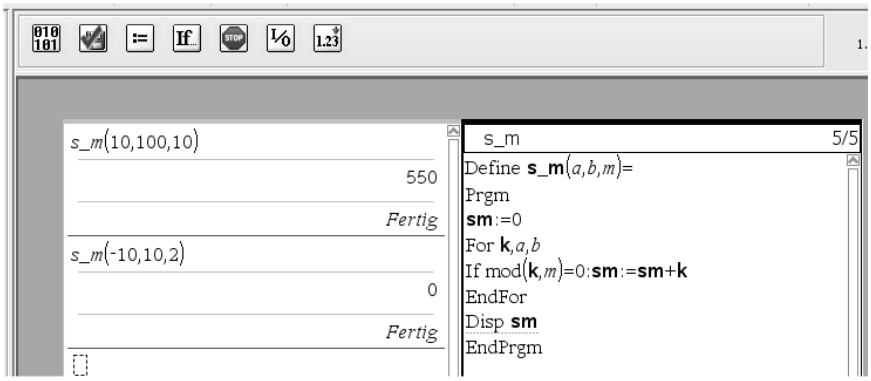
Kurzanleitung zum Programmieren mit dem TI-*Nspire*

Das erste Programm **s_m(a,b,m)** soll alle Zahlen k mit a ≤ k ≤ b, die durch m teilbar sind summieren und die Summe ausgeben.

Gehen Sie vom Calculatorfenster über die Programm-Editor-Schaltfläche in den Editor und schreiben Sie das kleine Programm **s_m**:





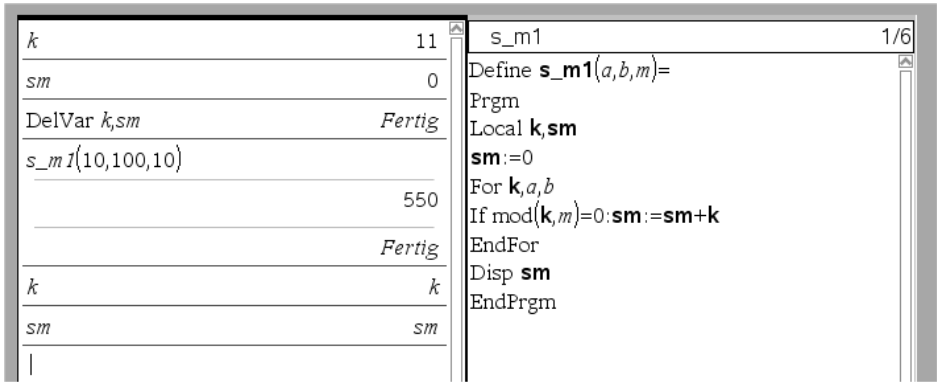


Im Rechenfenster wird das Programm getestet.

Bemerkenswert ist, dass nach dem ersten erfolgreichen Programmlauf die Laufvariable k im Editor fett ausgewiesen wird. Im Rechenfenster sehen wir, dass die Variable mit dem Wert b+Schrittweite der *for-next*-Schleife belegt wurde.

Wenn k als lokale Variable definiert wird, tritt das nicht ein. Wir werden bald nochmals darauf zurückkommen. Vergleichen Sie mit **s_m1**.

Ohne *Disp*-Answeisung erfolgt keine Ausgabe. Der Wert der Summe wird global gespeichert, da sm nicht als lokale Variable definiert wurde. In einem Programm können sowohl lokale als auch globale Variable vorkommen. Hier wäre es sinnvoll, zumindest k als lokal zu definieren. Ein Programm wird immer mit der Feststellung *Fertig* beendet.

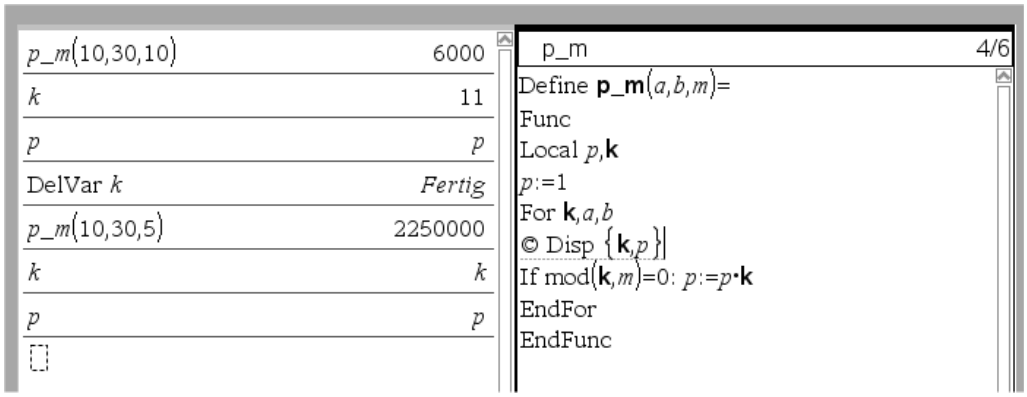


Zur Eingabe der Befehle dienen sowohl der Katalog als auch die Schaltfläche in der Programmierumgebung, in denen die am häufigsten verwendeten Befehle zu finden Sie. Sehen Sie sich die Menüs bitte an!

Vergessen Sie nicht nach jeder Änderung die aktuelle Version des Programms (der Funktion) mit dem grünen Häkchen auf syntaktische Fehler prüfen zu lassen und im Dokument zwischenzuspeichern. Speichern Sie bitte in regelmäßigen Abständen das ganze Dokument. Ein Programmabsturz kann die ganze Arbeit zunichte machen.

Das zweite Programm **p_m(a,b,m)** soll das Produkt aller Zahlen k mit a ≤ k ≤ b, die durch m teilbar sind bilden und ausgeben. Dieses Programm wird als Funktion definiert.

Der Einstieg erfolgt wie oben, nur der Typ ist *Funktion*. Ich rate, den Editor zu benützen, da mit ihm er wesentlich bequemer zu arbeiten ist, wie im Calculator.



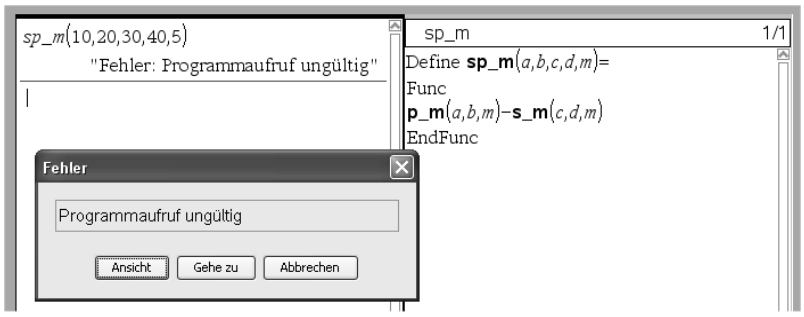
Hier wird ein Funktionswert berechnet und dieser ausgegeben, die *Disp*-Anweisung ist nicht nötig, wenn es nur um den letzten Wert geht. Die *Disp*-Anweisung ist aber oft sinnvoll, wenn auch Zwischenergebnisse ausgegeben werden sollen - etwa bei der Fehlersuche. Teste, was geschieht, wenn unmittelbar nach der *For*-Anweisung die Zeile disp {k,p} eingefügt wird.

Ganz wichtig ist, dass in einer Funktion alle auftretenden Variablen - und vor allem inkl. allfälliger Laufvariablen in Schleifen (wie hier k in der *for-next*-Schleife) als lokale Variable definiert werden müssen.

Vorerst erscheint hier das k noch fett gedruckt, da es von einer früheren Anwendung noch belegt ist. Nach *delvar k* im Rechenfenster, wird nach nochmaliger „Behandlung“ mit dem grünen Häkchen auch das k wieder unmarkiert auftreten. Diese nochmalige Behandlung ist nicht notwendig!

Wenn keine globalen Variablen benötigt werden, ist es zumeist ausreichend mit Funktionen zu arbeiten, mit diesen ist man auch viel flexibler, da die Ergebnisse sofort übernommen werden können.

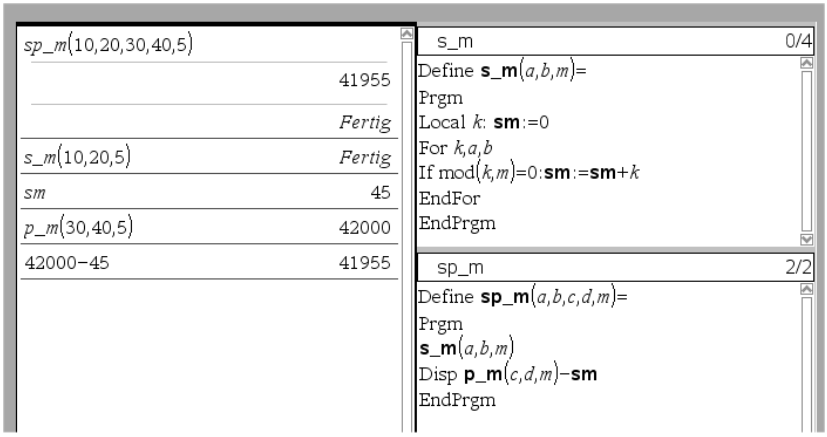
In einem Programm - oder wieder in einer Funktion - sollen sowohl **s_m**, als auch **p_m** aufgerufen werden. **sp_m(a,b,c,d,m)** soll die Differenz aus **p_m(a,b,m)** und **s_m(c,d,m)** bilden und ausgeben.



Im ersten Versuch erhalten wir eine Fehlermeldung! Nach *Gehe zu* wird die Zeile zwischen *Func* und *EndFunc* markiert.

Eine Funktion darf kein Programm als „Unterprogramm“ oder „Unterfunktion“ enthalten.

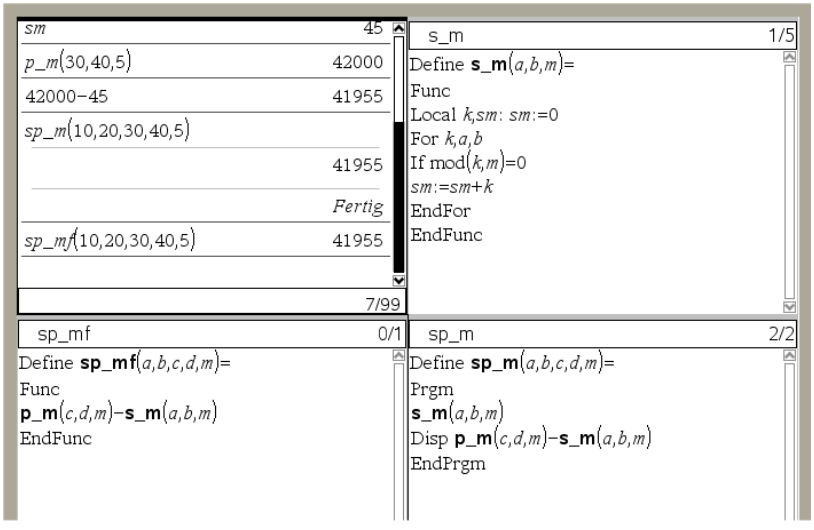
Wir müssen das Programm **s_m** so ändern, dass wir dessen Ergebnis übernehmen können. Zu diesem Zweck muss **s_m** sein Ergebnis in einer globalen Variablen (zB **sm**) ablegen. **s_m** muss im Hauptprogramm mit den gewünschten Parametern erst ausgeführt werden, dann kann man **sm** übernehmen.



Das ist sicher nicht der eleganteste Weg und lässt sich auch nicht leicht dokumentieren.

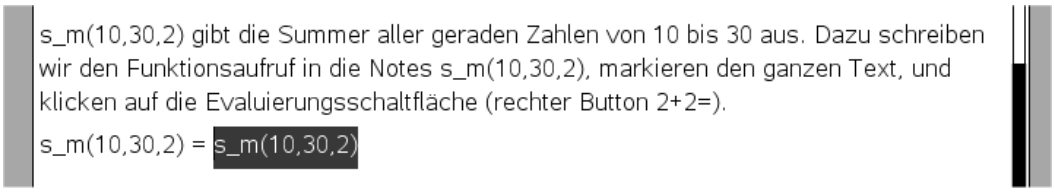
Am einfachsten ist es, auch **s_m** zu einer Funktion zu machen (dabei darf nicht auf die lokalen Variablen vergessen werden!) und dann beide Funktionen in einem zusammenfassenden Programm oder wieder in einer Funktion zu kombinieren.

Im nächsten Bild können Sie dann alles zusammen nochmals sehen.

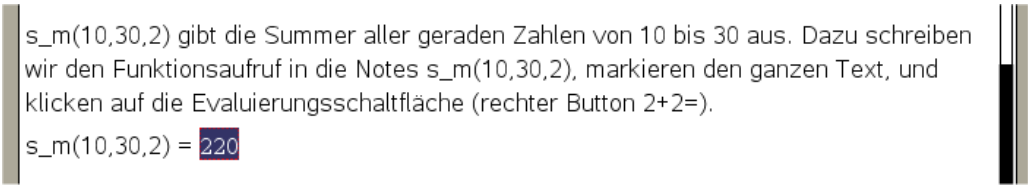


Diese Funktionen lassen sich aus allen anderen Applikationen auch aufrufen, zB aus den Notes.

So sieht es zB unmittelbar nach der Markierung aus:



und so nach der Auswertung:



Dem Aufbau einer eigenen Programmbibliothek am Beispiel Trigonometrie ist ein eigener Beitrag gewidmet, der auch auf dieser website zu finden ist.

Hinweis: ein Buch zum Programmieren mit dem Nspire-CAS wird demnächst bei bk-teachware erscheinen.

Josef Böhm
nojo.boehm@pgv.at