Lektion 4: Schleifen | **Übung 3: Die Loop-Schleife**

In dieser Übungseinheit wirst du die „Allzweckschleife“ **Loop…EndLoop** kennen lernen.

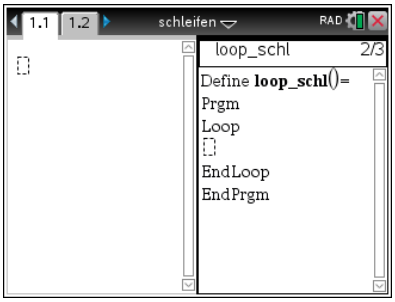
Lernziele:

- Die **Loop…EndLoop**-Schleife einsetzen
- Die **Exit**-Anweisung zum Verlassen einer Schleife verwenden

Hinweis: Die meisten traditionellen Programmierlehrer mögen die **Loop…EndLoop**-Schleife nicht, da sie zu schlechtem Programmierstil wie ‘Spaghetticode’ verleitet (Programme, die mit Hilfe von GoTo-Anweisungen wild herumspringen). Die **Loop…EndLoop**-Anweisung wird hier mit der Absicht vorgestellt, dass sie sinnvoll eingesetzt wird, nämlich unter Verwendung einer einzigen **Exit**-Anweisung und unter unbedingter Vermeidung von **GoTo**-Anweisungen!
Ein Vorteil der **Loop…EndLoop**-Struktur ist die *Möglichkeit* von mehreren Exits. Das sollte aber die Ausnahme sein, denn man diese Situation immer vermeiden. Die Exit-Anweisung zwingt das Programm, die erste Anweisung nach **EndLoop** auszuführen. Damit kann es keinen Zweifel darangeben, wo das Programm weitergeht.

Welche Schleife ist das, Loop…EndLoop?

Die **Loop…EndLoop**-Schleife gestattet eine flexiblere Anwendung des Schleifenkonzepts. Die Anweisungen im Schleifenkörper werden so lange ausgeführt, bis eine **Exit**-Anweisung innerhalb der Schleife zu ihrem Abbruch führt.



Wenn kein **Exit** angetroffen wird, führt das zu einer “Endlosschleife”. Wenn du unbeabsichtigt in eine Endlosschleife geraten bist, kannst am Handheld mit gedrückter und gehaltener on-Taste das Programm unterbrechen.

Eine **Loop…EndLoop**-Schleife wird zumindest einmal durchlaufen, da es für den Eintritt in sie keine Bedingung gibt.

Exit zwingt das Programm, mit der Anweisung, die dem **EndLoop** folgt, fortzufahren.

Beispiel: Wie oft muss man einen Würfel werfen, bis man zweimal hintereinander die gleiche Augenzahl erhält?

Beachte, dass im rechtsstehenden Programm die **Loop…EndLoop**-Schleife mit einem **Exit** unter einer *Bedingung* verlassen wird. Wenn der Zufallsgenerator *randInt*(1,6) hintereinander zwei gleiche Werte erzeugt, wird die Schleife über **Exit** verlassen und die **Disp**-Anweisung am Programmende ausgeführt.

```
loop_schl
Define loop_schl()=
Prgm
Local t,c,n
t:=randInt(1,6): c:=1
Loop
n:=randInt(1,6)
If t=n Then
  Exit
Else
  t:=n: c:=c+1
EndIf
EndLoop
Disp c," mal"
EndPrgm
```

Exit findest du über ***menu> Übertragungen> Exit***.

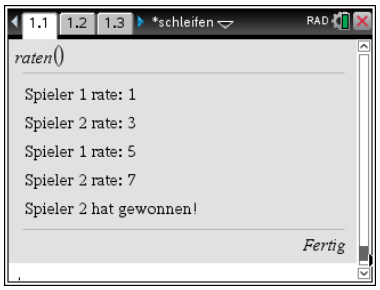
Das Programm lässt sich auch mit einer **While**-Schleife erzeugen.

Bemerkung zur Syntax: Beachte im Programmcode oben den Gebrauch des Doppelpunkts zur Trennung zweier Anweisungen in einer Zeile. Das erzeugt einen kürzeren und kompakteren Code.
Die Anweisung **Exit** findet man über ***menu> Übertragungen> Exit***.

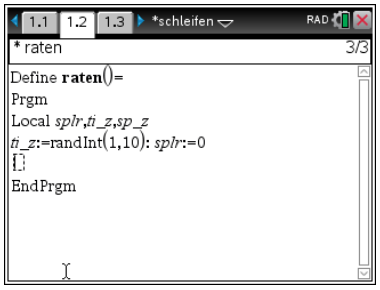
Hinweis: Ein Vorteil der **Loop…EndLoop**-Struktur ist die Möglichkeit, mehrere Exit-Punkte einbauen zu können. Das sollte aber mit Schülern so lange vermieden werden bis sie mehr Programmierkenntnisse haben, denn das verleitet zu unsauberem Programmieren.

Programm: Errate die Zahl!

Um den Gebrauch von **Loop…EndLoop** zu demonstrieren, wollen wir in Spiel für zwei Personen entwickeln. Eine ganze Zahl von 1 bis 10 soll erraten werden. Wenn ein Spieler die, vom Rechner erzeugte Zahl errät, wird das Spiel mit einer „Gewinnanzeige“ beendet. Ein möglicher Spielverlauf wird im rechten Screenshot gezeigt.



1. Beginne ein neues Programm und gib ihm den Namen **raten**.
2. Starte das Programm mit der Initialisierung des Spiels. Führe drei Variable ein: die Nummer des Spielers (*splr*), die vom Computer erzeugte Zufallszahl (*ti_z*) und die vom Spieler geratene Zahl (*sp_z*).



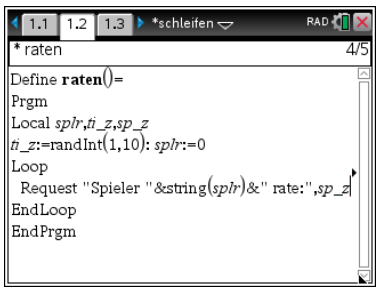
Der Computer erzeugt eine Zufallszahl zwischen 1 und 10. Wir starten mit dem Spieler 0. Das macht den Wechsel der Spielernummer einfacher, wie wir gleich sehen werden.

Hinweis: Denke daran, lokale Variable zu verwenden, um das aktuelle Problem nicht mit überflüssigen Variablen zu überladen.

3. Dann bilden wir die Schleife und fordern den ersten Spieler auf, die Zahl zu erraten. Die vollständige Anweisung dazu lautet:

   Request "Spieler " & string(*splr*) & " rate:", *sp_z*

   - das ‘**&**’ verkettet die Strings. Da die Eingabeaufforderung mit **Request** nur ein String sein kann, wird die Variable *splr* mit der **string()**-Funktion (aus dem Catalog) in eine Zeichenkette umgewandelt.

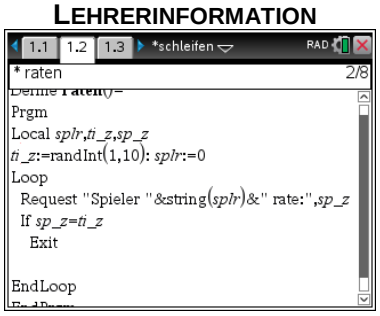


Hinweis: Rückblick: die Verkettung ist die Zusammenführung von zwei Zeichenketten zu einer, wobei die Zeichen der zweiten ans Ende der ersten angefügt werden.
Du musst die **string()**-Funktion verwenden, um einen numerischen Wert in einen String zu verwandeln, bevor du ihn mit einem String verketten kannst. **string( )** findet sich im Catalog.

4. Als nächstes bilde die **Exit**-Bedingung. Sobald ein Spieler die Zahl erraten hat, wird die Schleife verlassen. Dazu reicht die einfachste **If**-Anweisung:

   If sp_z = ti_z
   Exit

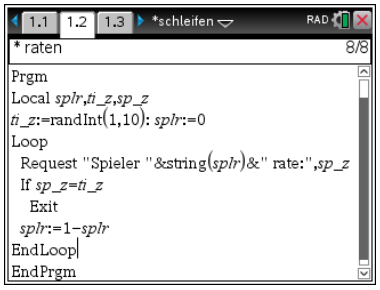
   Erinnere dich, dass ein **If** ohne **Then** die nächste Anweisung ausführen lässt, wenn die Bedingung erfüllt ist, anderenfalls wird sie übersprungen.



5. Für den Wechsel der Spieler gibt es eine kluge Anweisung:

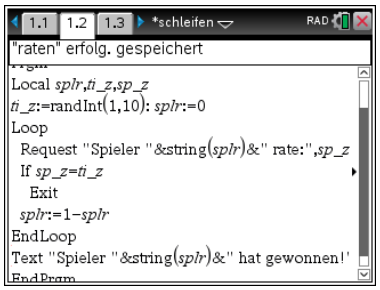
   splr* := 1 – *splr

   Diese kurze Anweisung schaltet den Wert von *splr* zwischen 0 und 1 hin und zurück. D.h., wenn *splr* = 0, dann wird auf 1 geschaltet, und wenn *splr* = 1, dann wird wieder auf 0 gestellt.



6. Zum Schluss füge mit einer Anweisung nach der Schleife eine Bemerkung für den Gewinner hinzu, wie etwa:

   Text “Spieler ” & string(*splr*) & “ hat gewonnen!”



Tipp: Wenn dir ‘Spieler 0’ und ‘Spieler 1’ nicht gefällt, dann addiere einfach 1 zur Variablen *splr* in dieser und in der **Request**-Anweisung. Am Display erscheinen dann 1 und 2, obwohl der Rechner 0 und 1 für die Spielernummern verwendet.

7. Vergiss nicht, das Programm vor jedem Testlauf zu speichern, denn du könntest ja in eine Endlosschleife geraten! Wenn das Programm einmal läuft, musst du spielen, bis ein Gewinner erscheint. Es gibt keinen Ausweg aus der **Request**-Anweisung. (Es gibt einen Trick: Gib keine Zahl ein!)

Hinweis: Wie bei jedem Dokument ist es auch bei einem Programm wichtig, regelmäßig zu speichern, so dass Änderungen nicht verloren gehen, wenn etwas falsch läuft!