LOOPS

START – NEXT

This is a simple loop, the syntax is like this:

InitialValue FinalValue

START

  [body of the loop]

NEXT

When the program reaches START, an internal variable is initialized to InitialValue Then the body of the loop is executed and when the NEXT is reached the internal variable increments in 1; if its value is less or equal than FinalValue the control return to START and the loop is executed again, if the internal variable value is greater than FinalValue the control exits the loop and continue to the next statement in the program after NEXT. If the FinalValue is less or equal than the InitialValue the loop is executed just once. Notice that you don't have access to the internal variable.

Example 1:

```
« 1 3
   START
     "RICK"
   NEXT
»
```

This program put the word RICK in level 1:, 2: and 3:.

Example 2:

```
« 1 2.5
   START
     "RICK"
   NEXT
»
```

This program put the word RICK in level 1: and 2:.

START – STEP

This is a more complex loop, the syntax is :

InitialValue FinalValue

START

  [body of the loop]

  StepValue

STEP

StepValue has to be right before STEP and can be a constant value or a value that is calculated each time the loop is executed, it also can be positive or negative. We have two cases here.

1. **InitialValue is less than FinalValue**: In this case StepValue has to be positive and the way this loop works is similar to the START-NEXT loop, except that when the STEP is reached, the internal variable is incremented in StepValue instead of 1. If StepValue is negative, the control exits the loop and continue to the next statement in the program.
2. **InitialValue is grater than FinalValue**: In this case StepValue has to be negative. When START, is reached, an internal variable is initialized to InitialValue Then the body of the loop is executed, then, when STEP is reached, the internal variable is decremented in StepValue, if its value is grater or equal than FinalValue the program control return to START and the loop is executed again, if the internal variable value is less than FinalValue, the control exits the loop and continue to the next statement in the program. If StepValue is positive, the control exits the loop and continue to the next statement in the program.

Note: If as result of a calculation or for whatever reason, the value of StepValue is always zero then the value of the internal variable never changes and the loop runs indefinitely; you have then a infinite loop.

You can try simple programs like:

Example 1:

```
« 2 6
  START
    "RICK"
    2
  STEP
»
```

and see how this works.

Example 2: The next sample generates a random number between 0 and 9 and in 10 iteration generates different random numbers (between 0 and 9), in each iteration the program compares if the new number generated is equal to the first number generated, if the numbers are equal the program tags the first number with the word "EQUAL" and exits the loop (by setting StepValue to -1) or else sets StepValue to 1 to continue the loop. At the end if none of the numbers generated in the iterations were equal to the first number generated, the program just shows the first number generated (with no tag).

```
«
 RAND 10 * IP                @ Generates the first number
 1 10
 START                       @ Starts a loop from 1 to 10
   IF RAND 10 * IP OVER ==   @ Generate another number and compares them
   THEN "EQUAL" →TAG -1      @ Tags the number and sets StepValue to -1
   ELSE 1                    @ Sets StepValue to 1
   END
 STEP                        @ End of the loop
»
```

FOR – NEXT

This is also a simple loop similar to START-NEXT loop, but now we have a variable that acts as a counter the name of that variable has to be right after the FOR, the syntax is like this:

InitialValue FinalValue

FOR

  CounterVariable

  [body of the loop]

NEXT

When the program reaches FOR, the CounterVariable is initialized to InitialValue Then the body of the loop is executed and when the NEXT is reached CounterVariable increments in 1 and if its value is less or equal than FinalValue the control return to FOR and the loop is executed again, if the CounterVariable value is greater than FinalValue the control exits the loop and continue to the next statement in the program after NEXT. If the FinalValue is less or equal than the InitialValue the loop is executed just once. CounterVariable is a local variable that exist and can be used only inside the loop.

Example: Given an integer n, this program find the sum of the first n integers, we are not going to use a variable to find this sum, instead, we are going to initialize a number to zero and keep it on the stack.

```
«
 0 1 ROT    @ Initalice the sum to 0, InitialValue to 1 and FinalValue to n
 FOR        @ Starts the loop
  j         @ Indicates that the name of CounterVariable is j
  j +       @ Recall the value of j and adds to sum
 NEXT       @ Ends the loop
»
```

This program does not validate your input.

FOR – STEP

This loop has the following syntax:

InitialValue FinalValue

FOR

  CounterVariable

  [body of the loop]

  StepValue

STEP

There is no much to say here, everything that was said about the START-STEP loop applies here with the difference that now we have a CounterVariable and we can use it in the body of the loop.

Example: The Maclaurin series of sin(x) is:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots$$

Given a real number as input, the next program find the Maclaurin series of sin(x) truncated to the first six terms, we are not going to use any variables here, notice that we need a sum and a number one that is going to switch from positive to negative in each iteration, let's call it SignValue.

```
«
 1 0                @ Sets SignValue and sum
 1. 11.             @ InitialValue and FinalValue are set as reals
 FOR
  j
  PICK3 j ^ j ! /   @ Calculate the j(th) term, x was in level 3:
  PICK3             @ Recall SignValue witch was in level 3: and
  * +               @   Multiply by j(th) termn and then add to sum
  SWAP NEG SWAP     @ swaps SignValue change its sign and swaps it back
  2                 @ StepValue is constant and set to 2
 STEP
 NIP NIP           @ Clear x and SignValue
»
```

For 1 this program returns:

.841470984648

and the function SIN (angle mode in RADians)in the calculator returns:

.841470984808

Do not forget that if for some reason, the value of StepValue is always zero then the value of CounterVariable never changes and the loop becomes infinite.

DO – UNTIL - END

This loop has the following syntax:

DO

 [body of the loop]

UNTIL

 TestStatement

END

This loop works like this: when the program control reaches DO the body of the loop is executed, once it reaches UNTIL the TestStatement is evaluated, when END is reached, if the evaluation of TestStatement is zero the program control goes to DO and repeats the loop, in other case the loop is ended and the control goes to the next statement after END. Since the test is performed at the end of the loop, the body of the loop is executed at least once. If the evaluation of TestStatement is always zero, you have an infinite loop.

Example: The next program find the square root of a real number using the Newton Method:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \qquad (1)$$

To find the square root of a number b, we have to solve the equation $f(x) = x^2-b = 0$. replacing f(x) in (1) and simplifying we have:

$$x_1 = x_0 - \frac{x_0^{\,2} - b}{2x_0};\ x_1 = (x_0 + \frac{b}{x_0})\frac{1}{2} \qquad (2)$$

This program uses the same number b as the initial guess and ends when x_1 and x_0 are equal in at least the first 9 significant digits.

```
«
 DUP
 DO
  DUP2 / OVER + 2. /   @ Calculate x1
 UNTIL
  SWAP -9 RND          @ x1 is now in level 2: and x0 in level 1: and round it
  OVER -9 RND          @ Gets a copy of x1 in level 1: and rounds it
  ==
 END
 NIP
»
```

For 1234567 this program returns:

1111.11070556

and the function √ in the calculator returns the same number.

WHILE – REPEAT - END

This loop has the following syntax:

WHILE

  TestStatement

REPEAT

  [body of the loop]

END

This loop works like this: when the program control reaches WHILE the TestStatement is evaluated, and then, REPEAT executes the body of the loop if TestStatement was evaluated to something different than zero and then the program control goes to WHILE and repeat the loop, if TestStatement was evaluated to zero the program control exits the loop and jumps to the statement after END. Since the test is performed at the beginning of the loop, the body of the loop is never executed if TestStatement evaluates to zero the first time. If the TestStatement is always evaluated to something different than zero, the loop is infinite.

Example: Given an integer n, the next program returns a list with integers from 1 to n in a random order. This program uses a START-NEXT loop from 1 to n and for each iteration, a WHILE-REPEAT-END loop generate a random number between 1 to n until it finds a number that is not in the list and then add it to the list.

```
«
 { } 1 PICK3
 START
  WHILE
   OVER RAND * CEIL  @ Genrates a random number  between 1 to n
   DUP2 POS          @ Returns 0 if the number wasn't found in the list
  REPEAT             @  Discard the number if POS found its position in the
   DROP              @   list
  END
  +                  @ If the number wasn't in the list, it is added the list
 NEXT
 NIP
»
```