hp calculators

HP 30b Programming – loops and subroutines

Programming on the HP 30b

Programming functions for loops and subroutines

Example programs using loops

- Count up and count down
- Finding the present value of a series of equal payments

Example programs using subroutines

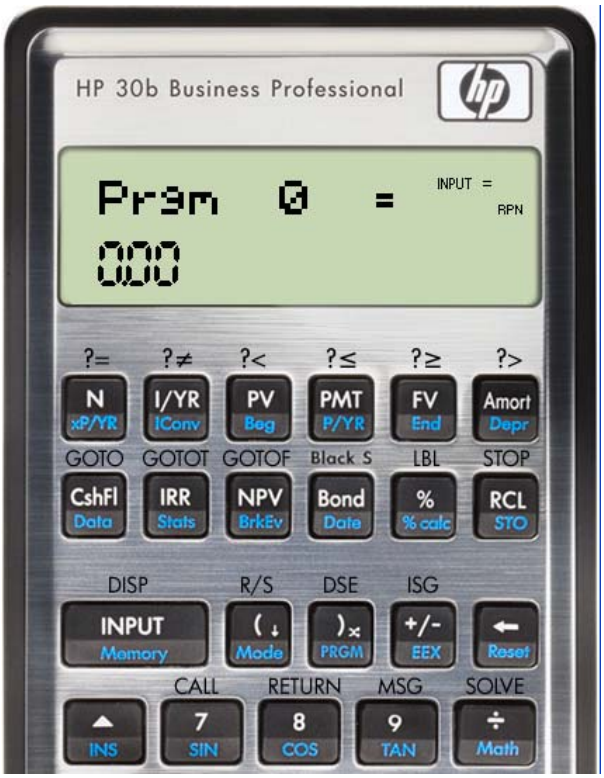
- Subroutines to allow RPN programs to run in algebraic or chain mode
- Quadratic equation root finder using subroutines

Programming on the HP 30b

The HP 30b Business Professional calculator includes a programming capability designed to help automate repetitive calculations and extend the usefulness of the built-in function set of the calculator. The capability includes the creation of up to 10 separate programs using up to 290 bytes of memory among them.

Programs record keystrokes, with each keystroke using one byte of memory, although some commands use more than one byte, as described later. In addition, many program-only functions are provided for conditional tests, conditional and unconditional "gotos", looping, displaying intermediate results and even calling other programs as subroutines.

This learning module will cover using loops and subroutines in the HP 30b programming environment in some detail. Other learning modules will show how to enter and edit programs, how to automate short, repetitive tasks, as well as showing several example programs to help get you started.



As shown in the picture at right, the HP 30b has additional functions assigned to the keys that are program-only functions. Other than the Black-Scholes function (shown as Black S), which is not a program function but a financial function, these functions are not printed or labeled on the actual HP 30b itself. However, an overlay is provided that lays over the top rows of keys that help indicate how these functions are mapped to the keys.

Each of these functions is inserted into a program by pressing the shift key and holding it down while pressing the key under which the program function is displayed. For example, to insert a LBL (label) command, press [shift] and, while holding it down, press [%]. In these learning modules describing programming, this will be shown as [shift] + [%]. Pressing that key combination will insert a LBL instruction into a program in program edit mode. Pressing that key combination in calculation mode will do nothing.

There are 10 numbered slots available for programs, numbered from 0 to 9. These are displayed in the program catalog which is viewed by pressing [shift] [PRGM]. In the image above, the program catalog is displayed, showing Prgm 0 or program 0. Pressing the [▲] or [▼] keys will scroll through the list of 10 programs. Pressing [INPUT] will enter the selected program, allowing you to view the program steps stored in that program slot or to change the program steps. To exit this program editing mode and return to the program catalog, press [shift] [PRGM]. To exit the program catalog and return to calculation mode, press [ON/CE].

When a program is displayed, a number will be shown below it indicating how many bytes are used. If the program name is shown in reverse video, then the program has been assigned to a key and can be executed by pressing the appropriate key combination, even when in calculation mode. This is shown in the image at right. When viewing a program in the program catalog, pressing [shift] [Reset] [INPUT] will delete the presently displayed program and return you to the calculation environment. To delete all programs, press [shift] [Reset] [▲] [▲] [INPUT] [INPUT] while in calculation mode.

At different places within a program, you can insert a Label (LBL) command. A label defines a location to which program control may be transferred. The HP 30b can handle up to 100 labels within the entire program memory. These labels are a two-digit numeric value from 00 to 99. No label can be used more than once, which makes each label a "global" label and defined only once within the global program memory space. If you attempt to enter a label that has already been used, a message saying "Exists!" will be briefly displayed.

Programming functions for loops and subroutines

This section lists the functions useful in programming loops and using subroutines on the HP 30b. The programming functions provided include 6 conditional tests, as shown in the table below. For conditionals, if the test returns a true result, a value of 1 is pushed to the stack in RPN mode. If a test returns a false result, a value of zero is pushed. This result can then be used by the "go to" commands described below to alter program flow based upon a test performed by the program.

Notes: 1) These commands test the value in Y against the value in X, which is the way HP's RPL-based graphing calculators work but the opposite of the way conditional tests have worked on past models. It is very important to notice this difference. 2) In RPN mode, the original contents of stack levels Z, Y and X are pushed one level higher into the stack when the conditional test returns a 0 or 1 as a result. The original contents of stack level T are lost when a conditional test is executed. If stack level T is needed, be sure to save it in a memory register before executing a conditional test. 3) In algebraic and chain modes, an = is required after the X value to perform the comparison. This equal will end all pending operations, so be sure that a conditional test in algebraic or chain mode is not performed with any pending operations.

Conditional tests

Symbol	Description
?=	Y equal to X? In RPN mode, usage is Y X ?= to compare the value in Y to the value in X. In algebraic or chain mode, usage is Y ?= X = to compare the value in Y to the value in X. If they are equal, pushes a 1 to X. Otherwise, pushes a 0 to X.
?≠	Y not equal to X? In RPN mode, usage is Y X ?≠ to compare the value in Y to the value in X. In algebraic or chain mode, usage is Y ?≠ X = to compare the value in Y to the value in X, with the final = required to perform the comparison. If they are not equal, pushes a 1 to X. Otherwise, pushes a 0 to X.
?<	Y less than X? In RPN mode, usage is Y X ?< to compare the value in Y to the value in X. In algebraic or chain mode, usage is Y ?< X = to compare the value in Y to the value in X, with the final = required to perform the comparison. If Y is less than X, pushes a 1 to X. Otherwise, pushes a 0 to X.
?≤	Y less than or equal to X? In RPN mode, usage is Y X ?≤ to compare the value in Y to the value in X. In algebraic or chain mode, usage is Y ?≤ X = to compare the value in Y to the value in X, with the final = required to perform the comparison. If Y is less than or equal to X, pushes a 1 to X. Otherwise, pushes a 0 to X.
?≥	Y greater than or equal to X? In RPN mode, usage is Y X ?≥ to compare the value in Y to the value in X. In algebraic or chain mode, usage is Y ?≥ X = to compare the value in Y to the value in X, with the final = required to perform the comparison. If Y is greater than or equal to X, pushes a 1 to X. Otherwise, pushes a 0 to X.
?>	Y greater than X? In RPN mode, usage is Y X ?> to compare the value in Y to the value in X. In algebraic or chain mode, usage is Y ?> X = to compare the value in Y to the value in X, with the final = required to perform the comparison. If Y is greater than X, pushes a 1 to X. Otherwise, pushes a 0 to X.

There are three “go to” commands provided for use by the HP 30b. A GOTO command simply transfers control to the specified label. The GOTOT and GOTOF commands evaluate the number on the stack to determine whether to transfer control to the specified label or to continue with the next program step. These commands are described in the table below.

NOTE: The GOTOT and GOTOF commands consume their arguments off the stack. If the stack were holding a 0 value, a program that executed a GOTOF command would not only jump to the specified label, but would drop the zero from the stack as well! If the stack were holding a non-zero value, a program that executed a GOTOT command would not only jump to the designated level, but would drop the non-zero value from the stack. The GOTO command does not consume an argument from the stack.

Goto functions

Function	Description
GOTO	Go to the label specified. This is an unconditional jump to the label location. Displayed as Gto.
GOTOT	Go to the label specified if the value in X is non-zero. A conditional test returns a result of 1 if the conditional is true, but the GOTOT command jumps to the specified label for ANY non-zero value. Displayed as GT.
GOTOF	Go to the label specified if the value in X is zero. A zero is a false result from a conditional. Displayed as GF.

There are two commands that allow programs to loop or repeat a series of steps until a certain condition is met. These are Increment and Skip if Greater (ISG) and Decrement and Skip if Equal (DSE). These are described in the table shown below. Although not required, these looping functions are often used with a “go to” instruction in order to loop. It is also acceptable to place any other instruction after the “loop” function. That instruction will either be executed or skipped.

Looping functions

Function	Description
DSE	Decrement and Skip if Equal. Takes a memory register from 0 to 9 as an argument. Evaluates the value contained in the register for the form ccccc.eeeii, where ccccc is the value of the counter, eee is the ending value, and ii is the increment value. It decreases the value of ccccc by the value of ii, stores the result back into the memory register, and then compares it to the value eee. If the new value for ccccc is equal to or less than eee, the next program step is skipped. The default value for ii is 1. For example, if the value in memory 0 is 15.00802, a DSE 0 command would subtract 2 from integer 15 portion of memory 0 and store the result back into memory 0. It would then compare the integer portion in memory register 0 (which is now 13) to the value 8. Since 13 is not equal to or less than 8, the next program step would be executed. For another example, if memory register 0 held the value 5.00400, the default value used for ii to decrease the memory register’s contents would be 1. A DSE 0 command would subtract 1 from the contents of memory 0, store result into memory register 0, and then since 4 is equal to the .004 portion of memory 0, the next program step would be skipped.

Looping functions

Function	Description
ISG	Increment and Skip if Greater. Takes a memory register from 0 to 9 as an argument. Evaluates the value contained in the register for the form ccccc.eeeii. It increases the value of ccccc by the value of ii, stores the result back into the memory register, and then compares it to the value eee. If the new value for ccccc is greater than eee, the next program step is skipped. The default value for ii is 1. For example, if the value in memory 1 is 5.00804, an ISG 1 instruction would add 4 to the integer portion 5 of memory 1 and store the result back into memory 1. It would then compare the integer portion in memory register 1 (which is now 9) to the value 8. Since 9 is greater than 8, the next program step would be skipped. For another example, if memory register 0 held the value -3.00000, the default value used for ii to increase the memory register's contents would be 1. The ISG 0 command would add 1 to the integer portion of memory 0, which is -3, and store the result back into memory register 0, and then since the updated value of -2 is not greater than the 000 portion of memory 0, the next program step would be not be skipped.

HP 30b programs can call and use subroutines from other program locations. A subroutine may be used if the series of program steps is repeated several times to save space or perhaps if a section of code has already been keyed in and tested, to allow for a saving of time by not having to key in the code again.

The HP 30b allows up to 4 levels of subroutines. Attempting to call a fifth subroutine level will result in an error. The two subroutine functions are described below.

Subroutine functions

Function	Description
CALL	Transfers control to the label specified as a subroutine. The subroutine label can be in the current program or in any other program. Uses 2 bytes of program memory.
RETURN	Returns control to the step after the CALL instruction.

Program checksums

Each program has a checksum that is displayed when the program is viewed in the program catalog. This checksum is based upon the sum of each byte's internal reference code value times the position of the byte from the end of the program.

Example programs using loops

Example 1: Write a program that will display the a counter from 1 to 10, going up one at a time and then will display a counter from 10 down to 1, going down one at a time.

While a brute force approach could work by writing a program that had steps such as 1 Disp1 2 Disp1 … 2 Disp1 1 Disp 1, let's use the loop and conditional test instructions. One such program is presented below, which will work in RPN, chain or algebraic modes. Other variations are possible.

Keys Pressed	Explanation
	Enters program mode and displays the last program previously viewed in the program catalog. If you wish to enter your program into a different program number in the catalog, press ▲ or ▼ until the program number you wish to use is displayed. Use Prgm location 3 which is assumed to be empty. Then press:
INPUT	Enters program edit mode and displays the first line of the program
1 . 0 1	Count from 1 to 10 by 1. Storing1.01 is the same as storing 1.01001.
STO 1	Use memory 1 as the loop counter.
+ % 5 0	Label 50 will be the loop.
RCL 1	Recall the counter. Depending on the FIX setting, decimal digits might be shown.
+ INPUT 2	Display it for 2 periods (each period is approximately 200 milliseconds).
+ +/- 1	Increment the contents of memory 1. If the integer value is greater than 10, skip the next step.
+ CshFl 5 0	Loop back to label 50. This is line 10 of the program.
1 0	Count from 10 to 1 by 1. Storing10 is the same as storing 10.00001.
STO 1	Use memory 1 as the loop counter.
+ % 5 1	Label 51 will be the loop.
RCL 1	Recall the counter. Depending on the FIX setting, decimal digits might be shown.
+ INPUT 2	Display it for 2 periods (each period is approximately 200 milliseconds).
+) 1	Decrement the contents of memory 1. If the integer value is equal to 0, skip the next step.
+ CshFl 5 1	Loop back to label 51.
+ RCL	Stop the program. This is line 19 of the program.
	Exits program edit mode and returns to the program catalog.

The program is 32 bytes in length and has a checksum of 101. Press = to run the program.

Example 2: Write a program that will find the present value of a monetary value received each period that is stored in memory 1, if interest is stored in memory 2 and the number of periods is stored in memory 3. Display the present value of each amount for a short pause within the loop. Enter the interest rate such that 5 would represent 5%.

This example will reproduce the present value of an annuity function from the time value of money keys. First, an RPN version.

Keys Pressed	Explanation
	Enters program mode and displays the last program previously viewed in the program catalog. If you wish to enter your program into a different program number in the catalog, press ▲ or ▼ until the program number you wish to use is displayed. Use Prgm location 4 which is assumed to be empty. Then press:
INPUT	Enters program edit mode and displays the first line of the program
0	Initializes the total at 0. Values will be computed and added together. Starts the sum at 0.
+ % 6 0	Label 60 will be the loop.
RCL 2	Recall the I/YR.
RCL P/YR	Recall the P/YR.
÷	Divide to get the interest rate per period.
1 0 0	Divide by 100.
÷	

Keys Pressed	Explanation
1 +	Add 1.
RCL 3	Recall the number of periods.
+/-	Make N negative
y^x	Raise (1 + I/YR / P/YR) to the negative N power.
RCL 1	Recall the periodic amount.
×	Multiply the amount by the present value factor.
+ INPUT 2	Display it for 2 periods (each period is approximately 200 milliseconds).
+	
+) 3	Decrement the contents of memory 3. If the integer value is greater than 0, skip the next step.
+ CshFl 6 0	Loop back to label 60.
+ RCL	Stop the program. This is line 21 of the program.
PRGM	Exits program edit mode and returns to the program catalog.

This program is 30 bytes in length and has a checksum of 137.

Run the program and find the present value of 100 per period for 10 months, with an interest rate of 5%, compounded monthly. Enter the values required by pressing:

1 0 0 STO 1
1 0 STO 3
5 STO 2
= =

The values that will be displayed are 95.93, 96.33, 96.73, 97.13, 97.54, 97.94, 98.35, 98.76, 99.12, and 99.59, with a total of 977.46. The decimal places shown may be different depending on your display settings.

Now the version that will work in algebraic or chain modes. Note that this version and the RPN version cannot both be in the calculator at the same time since they use the same label numbers.

Keys Pressed	Explanation
PRGM	Enters program mode and displays the last program previously viewed in the program catalog. If you wish to enter your program into a different program number in the catalog, press ▲ or ▼ until the program number you wish to use is displayed. Use Prgm location 4 which is assumed to be empty. Then press:
INPUT	Enters program edit mode and displays the first line of the program
+ % 6 0	Label 60 will be the loop.
((((	
RCL 2	Recall the I/YR
÷	and divide
RCL P/YR	by the P/YR to get the interest rate per period.
÷	
1 0 0)	Divide by 100.
+ 1)	Add 1.
y^x	Raise (1 + I/YR / P/YR) to the negative N power.
RCL 3	Recall the number of periods.
+/-)	Make N negative.
×	Multiply the present value factor

Keys Pressed	Explanation
RCL 1)	by the periodic amount.
+ INPUT 2	Display it for 2 periods (each period is approximately 200 milliseconds).
+	add to the present value of the next amount when the loop is executed.
+) 3	Decrement the contents of memory 3. If the integer value is greater than 0, skip the next step.
+ CshFl 6 0	Loop back to label 60.
0 =	add 0 to end the loop and complete the pending operation
+ RCL	Stop the program. This is line 31 of the program.
PRGM	Exits program edit mode and returns to the program catalog.

This program is 39 bytes in length and has a checksum of 146. This program is run the same way as the RPN program above and the results displayed are the same. Note that this program does the main calculation loop entirely within a set of parentheses. This allows each loop's result to be added together one result at a time and avoids issues with pending operations.

Example programs using subroutines

Example 3: Write a program that Call subroutines that will change the mode to RPN at the start and then back to chain mode when the program finishes.

If you want to use chain (or algebraic) mode for normal HP 30b operation, you can write a short program that you can insert as a subroutine call at the start of a program written for RPN, allowing the program to be run in RPN mode automatically and another subroutine call at the end that will restore the original mode. If you make the change to RPN mode subroutine label 99 and the change back to chain or algebraic mode subroutine label 98, then you can run RPN programs by inserting a CALL 99 as the first step and a CALL 98 as the last step. Note that these two subroutines use memories 8 and 9, which are then unavailable for use by other programs. Note that since these programs disturb the stack, this may require further manipulation of the inputs required for the RPN programs. These examples are for illustrative purposes only. This program is 30 bytes long and has a checksum of 225.

Keys Pressed	Explanation
PRGM	Enters program mode and displays the last program previously viewed in the program catalog. If you wish to enter your program into a different program number in the catalog, press ▲ or ▼ until the program number you wish to use is displayed. Then press:
INPUT	Enters program edit mode and displays the first line of the program
+ % 9 9	Label for subroutine to set RPN mode.
STO 9	Store the present number in the display into memory 9.
RCL Mode	Recall present mode.
STO 8	Save in memory 8 so that the original mode can be restored by label 98
3 1 4	314 will change the mode to RPN with the 3 in the hundred's place. The 14 is an invalid value and will be ignored.
STO Mode	Store the value into mode. RPN mode will be set. No other changes will be made to mode.
(	The HP 30b is in RPN mode at this point. Performs a roll down instruction.
RCL 9	Recalls the value in the display at the start of the subroutine.
+ 8	Inserts a Return command to end the subroutine.
+ % 9 8	Label for subroutine to restore original chain or algebraic mode.
STO 9	Store the present number in the display into memory 9.
(	The HP 30b is in RPN mode at this point. Performs a roll down instruction.
RCL 8	Recall original mode.

Keys Pressed	Explanation
[ ] STO [ ] Mode	Store the value into mode. Original mode saved from label 99 will be restored.
RCL 9	Recalls the value in the display at the start of the subroutine.
[ ] + 8	Inserts a Return command to end the subroutine.
[ ] PRGM	Exits program edit mode and returns to the program catalog.

Example 4: Write a program to solve a basic quadratic equation of the form $ax^2+bx+c=0$ with real roots, using a subroutine to evaluate the square root of the discriminant for both roots. The solution to the quadratic equation with real roots is given by the formula presented below.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

While not particularly efficient since HP Solve can handle quadratic equations, this program will illustrate the use of subroutines when a series of steps is repeated. An RPN and an algebraic version are presented. The programs do no error checking.

The RPN version is presented below. To use, store the value of a into memory 1, b into memory 2 and c into memory 3. When the program finishes, the two roots will be in stack levels X and Y.

Keys Pressed	Explanation
[ ] PRGM	Enters program mode and displays the last program previously viewed in the program catalog. If you wish to enter your program into a different program number in the catalog, press ▲ or ▼ until the program number you wish to use is displayed. Then press:
INPUT	Enters program edit mode and displays the first line of the program
[ ] + 7 8 8	Call subroutine 88 to compute the discriminant
RCL 2	Recall b
+/-	Make b negative.
) −	Swap X and Y and subtract
RCL 1	Recall a.
÷	
2 ÷	
[ ] + 7 8 8	Call subroutine 88 to compute the discriminant
RCL 2	Recall b
+/-	Make b negative.
+	Add
RCL 1	Recall a.
÷	
2 ÷	
[ ] + RCL	Inserts a Stop command to end the program. This is line 18 of the program.
[ ] + % 8 8	Label for subroutine to compute determinant.
RCL 1	Recall a.
RCL 3	Recall c.
× 4 ×	Compute 4ac
RCL 2	Recall b

Keys Pressed	Explanation
[ ] [x^2]	Square b.
[)] [−]	Swap X and Y and subtract
[ ] [√]	Swap X and Y and subtract
[ ] [+] [8]	Inserts a Return command to end the subroutine. This is line 30 of the program.
[ ] [PRGM]	Exits program edit mode and returns to the program catalog.

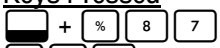
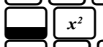
This program is 41 bytes in length and has a checksum of 174. Use this program to find the two roots of x^2-5x+6=0. Press the keys below to store the values for a, b and c:

[1] [ ] [STO] [1]
[5] [+/-] [ ] [STO] [2]
[6] [ ] [STO] [3]
[=] [=]

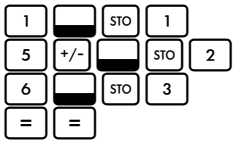
The program finishes its execution showing 3 in the display. Pressing [)] will display the 2nd root of 2.

Here is a version of this program that runs in algebraic or chain mode.

Keys Pressed	Explanation
[ ] [PRGM]	Enters program mode and displays the last program previously viewed in the program catalog. If you wish to enter your program into a different program number in the catalog, press [▲] or [▼] until the program number you wish to use is displayed. Then press:
[INPUT]	Enters program edit mode and displays the first line of the program
[(] [RCL] [2]	Recall b
[+/-]	Make b negative and
[−]	subtract the result of the subroutine called in the next step.
[ ] [+] [7] [8] [7]	Call subroutine 87 to compute the discriminant
[)]	and divide by the value of the denominator computed next.
[÷]	Recall a
[(] [RCL] [1]	and multiply by 2.
[×]	Inserts a R/S command to stop and display the first root.
[2] [)]	Recall b
[=]	Make b negative
[ ] [+] [(]	add to it the result from the subroutine called in the next step.
[(] [RCL] [2]	Call subroutine 87 to compute the discriminant.
[+/-]	and divide by the value of the denominator computed next.
[+]	Recall a.
[ ] [+] [7] [8] [7]	and multiply by 2.
[)]	Inserts a Stop command to end the program.
[÷]	
[(] [RCL] [1]	
[×]	
[2] [)]	
[=]	
[ ] [+] [RCL]	

Keys Pressed	Explanation
	Label for subroutine to compute determinant.
	Recall b
	Square b.
	Recall a and multiply
	by c and multiply by
	4.
	Compute the square root.
	Inserts a Return command to end the subroutine.
	Exits program edit mode and returns to the program catalog.

This program is 54 bytes in length and has a checksum of 010. Use this program to find the two roots of x^2-5x+6=0. Press the keys below to store the values for a, b and c:



The program stops showing the first root of 2 in the display. Pressing [shift] + [(] will resume execution and display the 2nd root of 3.

Conclusion

Other learning modules explain how to enter and edit programs, how to automate repetitive tasks, and how to use looping and subroutines. Still another module provides examples of already written programs to illustrate how HP 30b programming works. The programming capability of the HP 30b can provide a very helpful extension of the already powerful function set built into the HP 30b.