hp calculators

HP 30b Programming examples in RPN mode

Programming on the HP 30b

Example 1: Calculating digits of PI -
illustrates accessing data/cash flow memories
indirectly using memory register 0

Example 2: Finding prime factors of an integer

Example 3: Base conversions for bases 2-10

Example 4: Lunar lander game from the 1975
HP 25 Applications Program book

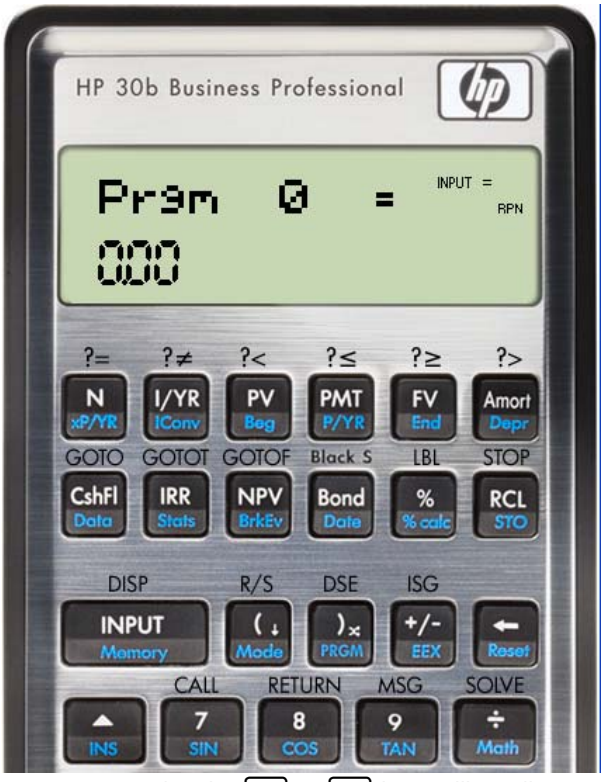
Example 5: Guess the secret number game

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. In addition, many program-only functions are provided for conditional tests, "gotos", looping, displaying intermediate results and even calling other programs as subroutines.

This learning module illustrates several example programs to help get you started. While these examples may not be directly related to what you need the HP 30b to do, they may illustrate how it can be extended by developing and using programs. These examples may stimulate programming ideas as well.

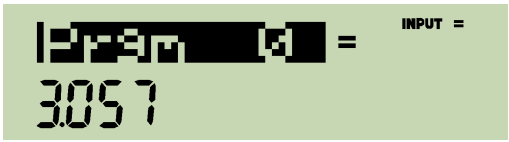


As shown in the picture at right, the HP 30b has additional functions assigned to the keys that are program-only functions. Other than the Black-Scholes function (shown as Black S), which is not a program function but a financial function, these functions are not printed or labeled on the actual HP 30b 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.

Example 1: Calculating digits of PI

The first example program will compute a user-specified number of digits of the constant PI and place the result in the cash flow / statistics data registers. It uses Euler’s convergence improvement applied to the Gregory series for PI.

This program illustrates an important feature of the HP 30b that is available to programmers: 100 data registers are available and can be accessed indirectly using data register 0 as an index or pointer. The statistics data registers begin from one end of this 100 register area and the cash flow values begin from the other end. They cannot overlap, so any values stored in one data area reduce the available number of registers for the other data area.

For example, if you press [5] [▬] [STO] [0] and then press [4] [▬] [STO] [▬] [Data]. The 5 will be stored in position 6 of the data registers, which is Y(3). It will be stored in the 6th position because the first position is referenced with an index of 0. To recall a value from the statistics data registers, store the proper index value into memory register 0 and press [RCL] [▬] [Data].

To use the cash flow registers, press [5] [▬] [STO] [0] and then press [4] [▬] [STO] [CshFl].The 5 will be stored in position 6 of the cash flow registers, which is #CF(2). It will be stored in the 6th position because the first position is referenced with an index of 0. To recall a value from the cash flow registers, store the proper index value into memory register 0 and press [RCL] [CshFl].

This allows for the use of two separate data areas of up to 100 total values, if a programmer wishes.

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 3 which is assumed to be empty. Then press:
[INPUT]	Enters program edit mode and displays the first line of the program
[▬] [STO] [9]	Save the number of registers to fill.
[RCL] [▬] [Mode]	Save the user’s mode to restore at end of program.
[▬] [STO] [5]	
[3] [0] [0]	
[▬] [STO] [▬] [Mode]	Set mode to RPN and fix 0 decimal places
[▬] [Reset]	Clear the cash flow registers
[▬] [+] [▼]	
[▬] [+] [▼]	
[INPUT] [INPUT]	
[▬] [Reset]	Clear the statistics registers. This provides maximum room for the statistics registers
[▬] [+] [▼]	
[▬] [+] [▼]	
[▬] [+] [▼]	where the results will be stored.
[INPUT] [INPUT]	

Keys Pressed	Explanation
RCL 9	Recall the number of registers to fill.
1 4 ×	Multiply by 14, which is 7 digits per register x 2 (which is the loop increment).
2 ▬ LN	
1 0 ▬ LN	Number of digits ÷ log base 2 of 10 are the number.
÷ ÷	of iterations needed (and the count down is by 2).
▬ Math	
▬ + ▲	
▬ + ▲	
INPUT	Accesses the IP (integer part) function in the math menu.
▬ STO 1	Saves the number of loops required in memory 1.
▬ + % 8 0	Label 80 is the top of the main loop.
RCL 9	
▬ EEX 3	
÷	
▬ STO 0	Set up the number of registers for ISG loop in label 81.
2	Initial value of term.
▬ STO ▬ Data	
0 INPUT	Initial value for carry.
▬ + % 8 1	Label 81 is the top of the loop through the statistics registers.
▬ EEX 7	
×	
RCL ▬ Data	
RCL × 1	
+	Numerator is Data(i)*n + carry * 10^7.
▬ STO 2	Set up the number of registers for ISG loop in label 81.
RCL 1	
2 ×	
1 +	Denominator is 2n+1.
▬ STO 3	
÷	
▬ Math	
▬ + ▲	
▬ + ▲	
INPUT	Accesses the IP (integer part) function in the math menu.
▬ STO ▬ Data	IP(Data(i)*n+carry*10^7)/(2n+1)) stored into Data(i).
RCL × 3	
RCL 2	
)	Performs a swap of X and Y, since this program is in RPN mode.
−	Carry into the next register.
▬ + +/- 0	Inserts ISG 0. Checks for the end of the statistics register loop.
▬ + CshFl 8 1	If not the end, loop back to label 81.
0	These next lines are needed because of the way the loop ends. A 2 is needed in Data(0).
▬ STO 0	
2	

Keys Pressed	Explanation
[ ] STO [ ] Data [ ] + [ )] [1]	Inserts DSE 1. Checks for the end of the term loop. If you add these two instructions BEFORE the DSE 1 at this step, you will see a "countdown" displayed as the program executes this loop: [RCL] [1] and [ ] + [INPUT] [1]. This can provide useful feedback on longer execution times.
[ ] + [CshFl] [8] [0] [RCL] [9]	If not the end, loop back to label 80. This is the final "fix up" loop. Just one pass through the registers to adjust the overflows.
[ ] STO [0]	
[0]	Initial value of the carry.
[INPUT]	
[ ] + [%] [8] [2] [RCL] [ ] Data	Label 82 is the top of the loop through the statistics registers to adjust for any overflow.
[+]	
[ ] STO [ ] Data	Add carry to register.
[ ] EEX [7]	
[ ] + [PV]	Inserts ?< conditional test. If true, no overflow.
[ ] + [IRR] [8] [3]	If true, skip over adjustment.
[RCL] [ ] Data	
[ ] EEX [7]	
[–]	Back out overflow.
[ ] STO [ ] Data	
[1]	Carry into next register.
[INPUT]	
[ ] + [CshFl] [8] [4]	
[ ] + [%] [8] [3]	Label 83.
[0]	No carry.
[INPUT]	
[ ] + [%] [8] [4]	
[ ] + [ )] [0]	Inserts DSE 0. Decreases register pointer and loops until all have been checked.
[ ] + [CshFl] [8] [2]	
[RCL] [ ] Data	
[+]	
[ ] STO [ ] Data	
[RCL] [5]	Loop over. Clean up by restoring user's original mode settings.
[ ] STO [ ] Mode	
[ ] Data	Program will end showing the cash flow registers to allow for review.
[ ] + [RCL]	Inserts Stop. Program over.
[ ] [ )] PRGM	Exits program edit mode and returns to the program catalog.

This program takes 154 bytes and has a checksum of 189. This program uses over half of the available program memory on the HP 30b.

To execute this program, enter the number of registers you wish to use for the results and press [=]. The first register will always contain the integer value of PI: a value of 3. The registers after the first one contain the decimal digits of PI, shown as an integer. Entering 1 to use registers 0 and 1 for storage will compute 7 decimal digits of PI in about 1 second while a value of 5 (using registers 0 through 5) will compute 35 digits in just a few seconds. The maximum number of registers that can be used is 99, which uses registers 0 through 99, for 693 digits of PI in under an hour. Also note that leading zeroes are not shown in the data registers. If the seven digits should be 0000023, the data register would simply show 23. The user must note and add any leading zeroes. If run with 5 as the number of registers to be used, the program ends with the following displayed. Press [▾] to see additional results as shown below.

To 35 decimals, the value of PI is 3.14159265358979323846264338327950288.

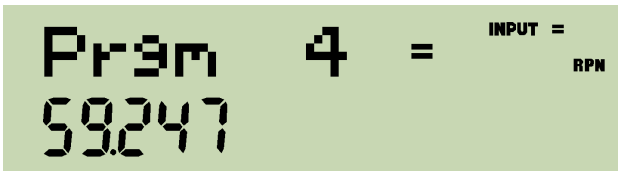
Example 2: Finding prime factors of an integer

Don would like to develop a program to factor some numbers into their prime factors. This example program will find the prime factors of an integer. For example, the number 10 can be factored into the product of two primes, 2 and 5. The number 13 is prime, as it can only be factored into 1 and 13.

Given a number, this program will return a series of prime factors. After each factor is returned, press [ ] + [( ] (which executes a R/S command) to continue the factoring of the number. If the original number is displayed, then the original number is prime. The program presented below MUST be run in RPN 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. 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.
[ ] [STO] [0] [2]	Store number to be factored in memory 0.
[ ] [STO] [1] [1]	Memory 1 stores the trial factor to use. Start with 2.
[ ] [STO] [2]	Memory 2 stores the increment to the trial factor. Starts with 1 to make 2nd factor tried equal to 3, then 2 to try 5, 7, 9…
[ ] + [%] [0] [0] [RCL] [0] [RCL] [1] [÷] [ ] [Math] [ ] + [▲]	Label 00 is the main loop.
[INPUT]	Accesses the FP (fractional part) function in the math menu. If 0, found a factor in memory 1.
[ ] + [NPV] [0] [2]	Inserts a Goto False command. If the result of the FP instruction is zero, go to label 02.
[RCL] [2]	Fractional part was non-zero. Number in memory 1 is not a factor. Increment factor to try next by recalling value in memory 2 and adding it to value in memory 1.
[ ] [STO] [+] [1] [2]	
[ ] [STO] [2] [RCL] [1]	These steps ensure the factor increment is 2, since the loop starts with this at a value of 1.
[ ] [x^2]	Trial factor squared. If larger than number being factored, stop the loop.
[RCL] [0]	Number being factored.
[ ] + [PMT]	Inserts a ?<= conditional test. If the value of memory 1, squared, is less than memory 0, places a 1 in the X register. Otherwise, places a 0 in the X register.
[ ] + [IRR] [0] [0] [1]	Inserts a Goto True command. If X is not equal to 0, go to label 00.
[RCL] [0]	Compare last factor found to 1.
[ ] + [N]	Inserts a ?= conditional test. If the value of memory 0 is equal to 1, places a 1 in the X register. Otherwise, places a 0 in the X register.
[ ] + [IRR] [0] [1]	Inserts a Goto True command. If X is not equal to 0, go to label 01.
[ ] + [( ]	Inserts R/S command and displays the present prime factor.

Keys Pressed	Explanation
[↓] + [%] [0] [1] [0] [=]	Label 01 is the destination if the last factor was 1.
[↓] + [RCL]	Inserts a Stop command and displays a 0. Indicates all prime factors have been found
[↓] + [%] [0] [2]	Label 02 indicates a prime factor was found.
[RCL] [1]	Display factor found.
[↓] + [(]	Inserts a R/S command and displays the current factor.
[↓] [STO] [÷] [0]	Update new number to factor by dividing number by factor found.
[↓] + [CshFl] [0] [0]	Inserts a Goto 00 command. Continues the loop.
[↓] [)]	Exits program edit mode and returns to the program catalog.

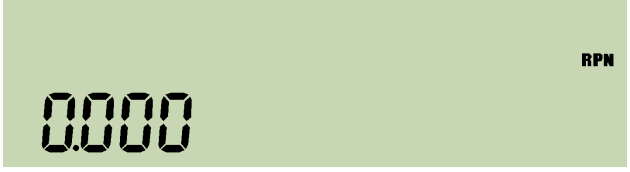


This program takes 59 bytes and has a checksum of 247. To execute this program from the program catalog, enter the number you wish to factor and press [=] [=]. If you have left the program catalog, reenter it by pressing [↓] [)].

Question 1: What are the prime factors of 55? Key in 55 and press [=] [=].

5 is displayed as the first factor found. Press [↓] + [(] to continue.

11 is displayed as the next factor found. Press [↓] + [(] to continue.



0 is displayed. This indicates the factors have been found. The prime factors of 55 are 5 and 11.

Question 2: What are the prime factors of 9999999967? Enter the program catalog by pressing pressing [↓] [PRGM]. Key 9999999967 and press [=] [=]. Be aware that this will take several minutes to run.

The original number 9999999967 is displayed as the first factor found. Press [↓] + [(] to continue.

0 is displayed. This indicates the factors have been found. 9999999967 is prime.

Historical note: The HP user club (not associated with Hewlett Packard) that came to be known as PPC published a journal for many years that included programs written by users. One such program was a "Speedy Factor Finder". The value used as a test case for speed improvements was the largest 10-digit prime number, 9999999967. This number proved prime using a program written for the HP 67 calculator in just under 3 hours.

Example 3: Base conversions

This example program converts a number from a base to another base in the range of bases 2 through 10, provided that one of the bases is in fact 10. For example, this program can convert from base 10 to a base 2 through 9, or can convert from a base 2 through 9 to base 10. To convert a number in base 8 to base 2, for example, you must perform an intermediate step by first converting the base 8 number to base 10 and then converting the resulting base 10 number to base 2. Bases greater than 10 are not supported by this program.

Inputs to this program are the number to convert, the input base, and the output base. The program presented below MUST be run in RPN 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. Use Prgm location 5 which is assumed to be empty. Then press:
[INPUT]	Enters program edit mode and displays the first line of the program.
[↓] [STO] [2]	Store the output base in memory 2.
[(]	This executes a roll down of the 4-level stack.
[↓] [STO] [1]	Store the input base in memory 1.
[(]	This executes a roll down of the 4-level stack.
[↓] [STO] [0]	Store the number to convert in memory 0.
[0]	
[↓] [STO] [3]	Initialize the output number in memory 3.
[1]	

Keys Pressed	Explanation
STO 4	Initialize the multiplier to use in memory 4.
+ % 1 0	Label 10 is the main loop.
RCL 0	
RCL 2	
÷	Divide number to convert by output base.
STO 0	Store result back into memory 0.
Math	
+ ▲	
INPUT	Accesses the FP (fractional part) function in the math menu.
RCL × 2	Multiply fractional part by output base to get digit.
RCL × 4	Multiply by digit position multiplier and
STO + 3	add to accumulated total.
RCL 1	Update multiplier by multiplying by
STO × 4	the input base.
RCL 0	
Math	
+ ▲	
+ ▲	
INPUT	Accesses the IP (integer part) function in the math menu. Takes the integer part of the earlier computed quotient.
STO 0	Store the result back into memory 0.
+ IRR 1 0	Inserts a Goto True command. If the integer part is not equal to zero, go to label 10. This will continue the loop until the quotient is zero.
RCL 3	Recall final output number in new base.
+ RCL	Inserts a Stop command. Program ends execution.
PRGM	Exits program edit mode and returns to the program catalog.

This program takes 54 bytes and has a checksum of 155. To execute this program from the program catalog, enter the number you wish to convert, press [=], enter the number's present base, [=], and enter the base you wish to convert it to and press [=] [=].

Convert 175 base 8 to base 10. Key in [1] [7] [5] [=] [8] [=] [1] [0] [=] [=].

175 base 8 is equal to 125 base 10. Now convert this result to base 2. Since when executed, the program leaves the program catalog, to run it again press: PRGM [1] [2] [5] [=] [1] [0] [=] [2] [=] [=].

175 base 8 is equal to 125 base 10 which is equal to 1111101 base 2.

Example 4: Lunar lander game

This example program simulates landing on the moon. It was originally published by Hewlett Packard in 1975 and can be found in the HP 25 Applications Program book.

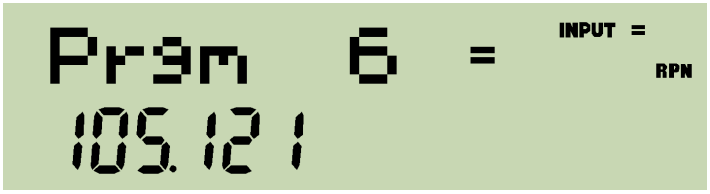
The game starts off with the rocket descending at a velocity of 50 feet/sec from a height of 500 feet. The velocity and height are shown in a combined display as -50.0500, the height appearing to the right of the decimal point and the velocity to the left, with a negative sign on the velocity to indicate downward motion. If a velocity is ever displayed with no fractional part, for example, -15, it means that you have crashed at a speed of 15 feet/sec. In game terms, this means that you have lost; in real-life, it signifies an even less favorable outcome.

You will start the game with 120 units of fuel. You may burn as much or as little of your available fuel as you wish (as long as it is an integer value) at each step of your descent; burns of zero are quite common. A burn of 5 units will just cancel gravity and hold your speed constant. Any burn over 5 will act to change your speed in an upward direction. You must take care, however, not to burn more fuel than you have; for if you do, no burn at all will take place, and you will free-fall to your doom! The final velocity shown will be your impact velocity. Any impact velocity over 5 feet/sec would probably doom your attempt. You may display your remaining fuel at any time by recalling memory 2.

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 6 which is assumed to be empty. Note: If you have the previous examples in memory, you will have to delete one of them before entering this program.
INPUT	Enters program edit mode and displays the first line of the program.
5 0 0 STO 0	Store the initial height in memory 0.
5 0 +/- STO 1	Store the initial downward velocity in memory 1.
1 2 0 STO 2	Store the initial fuel in memory 2.
3 0 4	This executes a roll down of the 4-level stack.
STO Mode	Set mode to RPN and 4 decimal places shown.
+ % 3 0 RCL 0 EEX 4	Label 30 is the main loop.
÷	Divide height by 10,000.

Keys Pressed	Explanation
RCL 1	
0	Compare velocity to 0.
+ PV	Inserts a ?< conditional test. If the velocity is less than 0, places a 0 in the X register. Otherwise, places a 1 in the X register.
+ IRR 3 1	Inserts a Goto True command. If X is equal to 0, go to label 31. Label 31 is when velocity is negative.
(	These steps are performed when velocity is positive. Performs a stack roll down.
+	Adds velocity to fraction displaying height.
+ CshFl 3 3	Inserts a Goto command. Jumps to label 33.
+ % 3 1	Label 31. These steps are performed when velocity is negative.
(	Performs a stack roll down.
)	Performs a stack swap of the X and Y registers.
−	Subtracts a negative velocity from a positive height.
+ % 3 3	Label 33. Destination after alternate paths for positive or negative velocity.
+ (	Inserts R/S command and displays V.X, velocity.height
RCL 2	
+ Amort	Inserts a ?> conditional test. If the input burn is greater than amount of fuel, prepare to crash.
+ IRR 3 4	Inserts a Goto True command. If X is equal to 0, go to label 34 and prepare to crash.
(	Performs a stack roll down. Burn is less than total fuel. Update acceleration, velocity, and height.
STO − 2	Subtract burn from fuel.
5 −	5 units cancels effects of gravity, so acceleration is burn minus 5.
STO 3	Store acceleration into memory 3.
2	
÷	
RCL 0	
+	
RCL 1	
+	New height = original height plus velocity plus acceleration.
STO 0	Store new height into memory 0.
0	Compare height to 0.
+ PV	Inserts a ?< conditional test. If the height is less than 0, places a 0 in the X register. Otherwise, places a 1 in the X register.
+ IRR 3 5	Inserts a Goto True command. If X is equal to 0, go to label 35. Label 35 represents a crash with fuel remaining.
(	Have not crashed. Performs a stack roll down.
RCL 3	Recall acceleration.
STO + 1	Store new velocity into memory 1.
+ CshFl 3 0	Inserts a Goto command. Jumps to label 30 to begin another loop.
+ % 3 4	Label 34. Determines velocity of crash with no fuel remaining.
RCL 3	Compute crash velocity as square root of (V^2 + 2gHeight), where g=5
x^2	

Keys Pressed	Explanation
RCL 0 1 0 × ↓ +/- ↓ STO 1	Display as a negative number to indicate a crash.
↓ + % 3 5	Label 35. Create display to indicate a crash occurred..
3 0 0 ↓ STO ↓ Mode	Set mode to RPN and 0 decimal places shown for a crash.
RCL 1	Recall crash velocity.
↓ + RCL	Inserts a Stop command. Program ends execution.
↓ PRGM	Exits program edit mode and returns to the program catalog.



This program takes 105 bytes and has a checksum of 121. To run this program from the program catalog, press =.

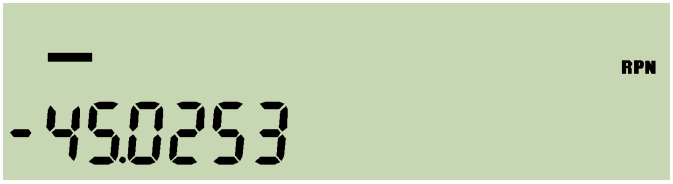
The initial descent display is shown. The landing craft is 500 feet high and descending at 50 feet / sec. Burn 5 units of fuel by pressing 5 ↓ + (.

Burn 0 units of fuel by pressing 0 ↓ + (.

Burn 10 units of fuel by pressing 1 0 ↓ + (.

Burn 10 units of fuel by pressing [1] [0] [▬] + [(].

Burn 5 units of fuel by pressing [5] [▬] + [(].



Burn 0 units of fuel by pressing [0] [▬] + [(].

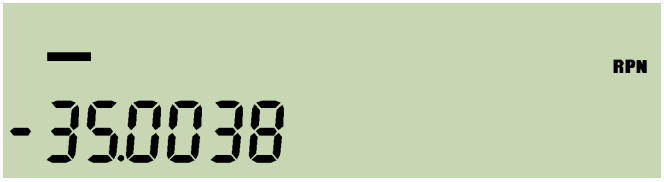
Burn 10 units of fuel by pressing [1] [0] [▬] + [(].

Check remaining fuel by pressing [RCL] [2].

Burn 10 units of fuel by pressing [1] [0] [▬] + [(].

Burn 5 units of fuel by pressing [5] [▬] + [(].

Burn 10 units of fuel by pressing [1] [0] [▬] + [(].



Burn 10 units of fuel by pressing [1] [0] [▬] + [(].

Burn 15 units of fuel by pressing [1] [5] [▬] + [(].

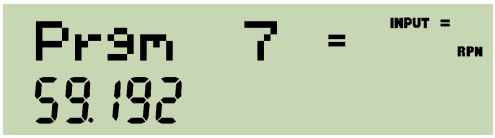
This is a crash. Perhaps you can do better?

Example 5: Guess the secret number game

This program generates a secret number between 0 and 99. The user enters a guess and the program indicates whether the guess is too high or too low. This looping process continues until you guess the number. By making proper guesses, any number can be found in 7 or fewer attempts.

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 7 which is assumed to be empty. Then press:
INPUT	Enters program edit mode and displays the first line of the program.
RAND	Get random seed.
EEX 2	Multiply by 100.
×	
Math	
+ ▲	
+ ▲ =	Accesses the integer part (IP) function from Math menu. Displayed as “Math Up Up =”
STO 0	
0	
STO 1	Initialize guess counter at 0.
+ % 7 0	Label 70. Main loop of program.
+ (	Inserts a R/S command. Enter your guess between 0 and 99.
=	Necessary to terminate digit entry of guess.
1	
STO + 1	Increment guess counter.
(	Roll the stack down.
RCL 0	
)	Swap. Puts your guess in X and secret number in Y.
+ I/YR	Inserts a not equal conditional test.
+ IRR 7 1	Inserts a Goto True command. If the guess is not the secret number, go to label 71.
RCL 1	Recall guess count.
+ 9	Insert a MSG command to display “Yes”
+ ▲	
+ ▲	
▲ ▲ ▲ ▲	
INPUT	Inserts a Y.
▲ ▲ ▲ ▲	
INPUT	Inserts an e.
+ ▲	
+ ▲	
▼ ▼	Inserts an s.
ON/CE	Terminates MSG character entry.
+ RCL	Inserts a Stop command. Game over. Number of guesses is in display.
+ % 7 1	Your guess was wrong.
+ Amort	Inserts ?> conditional test.
+ IRR 7 2	Inserts a Goto True command. If the secret number is greater than the guess, go to label 72.

Keys Pressed	Explanation
[ ] + [9]	Insert a MSG command to display “High”
[ ] + [▲]	
[▼] [▼] [▼]	
[INPUT]	Inserts an H.
[ ] + [▲]	
[▼] [▼]	
[INPUT]	Inserts an i.
[ ] + [▲]	
[▼] [▼] [▼] [▼]	
[INPUT]	Inserts a g.
[ ] + [▲]	
[▼] [▼] [▼]	Inserts an h.
[ON/CE]	Terminates MSG character entry.
[ ] + [CshFl] [7] [0]	Go back for another guess.
[ ] + [%] [7] [2]	Your guess was too low.
[ ] + [9]	Insert a MSG command to display “Low”
[ ] + [▲]	
[▲]	
[INPUT]	Inserts an L.
[ ] + [▲]	
[▲] [▲] [▲] [▲]	
[INPUT]	Inserts an o.
[ ] + [▲]	
[ ] + [▲]	
[▲] [▲]	Inserts a w.
[ON/CE]	Terminates MSG character entry.
[ ] + [CshFl] [7] [0]	Go back for another guess. This is line 32 of the program.
[ ] [PRGM]	Exits program edit mode and returns to the program catalog.



This program takes 59 bytes and has a checksum of 192. To play the game, press [=] while in the program catalog. Enter a guess of 40. [4] [0] [ ] + [(]. Note: since the number generated will be random, the game play illustrated below will probably not match your own experience, since a different secret number will probably be generated.

40 is too low. Enter a guess of 90. [9] [0] [ ] + [(].

90 is also too low. Enter a guess of 97. [9] [7] [▬] + [(].

97 is too high. Enter a guess of 95. [9] [5] [▬] + [(].

95 is too high. Enter a guess of 92. [9] [2] [▬] + [(].

The secret number was 92 and was found in 5 guesses!