Programming HP calculators

The only sample I have is an HP48SX calculator but I expect the syntax is similar for most programable HP calculators.

Preliminaries. To start editing a new program press the orange control key then press the (-) key to get matching braces

<< >>

The program is then typed in between these matching braces. It is helpful to type in your program in short readable lines and you can insert new lines as you go by pressing the blue control key followed by the left pointing blue arrow (←) which is over the period (-) key. When you have finished writing your program use one of the cursor control keys to move below the final brace pair >>. Then type the name you want to give the program inside single quotes (the key which produces a pair of single quotes is under the names up home). Finally press the key STO to store the program. If the syntax of your program is not correct the calculator will complain at this point and you can begin beating your head against the wall trying to find out what is wrong. Once you have successfully entered a program, the name of the program (or a truncation of that name) will appear along the bottom of the display when you hit the VAR key. You may need to use the NXT key to see the whole list.

To start editing a program that is already stored in the calculator press the single quote button (it is underneath up home) then enter the name of the program you want to edit between the single quotes. You can either do this by typing the name in or you can hit the VAR key and then press the white key under the appropriate program name. Next press enter to put this quoted name in the first stack position. Now press the blue control key followed by visit. This should load the program into memory for editing. Once you have finished editing it is only necessary to press ENTER to store the changes in a program of the same name.

Programming devices. Often the first thing one would like to do in writing an interactive program is to initialize a collection of variables that the program will compute with. Here is a scheme for doing this with an HP48

"X =?" " INPUT OBJ→
'X' STO

The first line in quotes will show up on the upper screen of the calculator when you run the program. The second empty " " quote is put in because INPUT requires two arguments. You can put a command in this slot but this is not necessary for entering numerical data. When the program hits the INPUT command it stops execution.
of the program and waits for you to key in a number and then press ENTER to indicate you are finished. Once the program resumes execution it hits OBJ→. This strange command is there because INPUT automatically accepts whatever you give it and stores it as a string in the first stack position. OBJ→ converts a string object, like “24.5” to the number 24.5. The next combination ‘X’ STO stores whatever is in the first stack position in the variable named X. The variable X created this way is a global variable. Incidentally, I spent a long time trying to figure out a way to interactively enter local variables. I discovered a way to do this but the technique necessarily involved subroutines that killed off the useful scope of the local variables! My suggestion is not to bother with local variables at all. Later we will deal with the nuisance associated with global variables.

Once one has variables with associated numerical values, one wants to be able to calculate with them and store the result in some place for future reference. Here is one way to do this on the HP48SX.

‘\( \sin(X + Y) \)’ →NUM ‘U’ STO

This takes the expression ‘\( \sin(X + Y) \)’ then substitutes the numerical values of X and Y into this expression to find a numerical result (that is what →NUM does—the HP48SX can do “symbolic calculations” so it needs to know that you want a number when you write \( \sin(X + Y) \)). Be careful with spaces—the arrow must be adjacent to NUM but there should be spaces between the other parts. Finally the resulting number is stored in the variable named U. If a “local variable” named U does not already exist this will create a “global variable” named U.

The only other ingredient we need to be able to write a useful program is a “do loop”. There are many ways to write such loops on this calculator but one simple syntax is

\[
<< 1 \ N \ FOR \ I \ldots \ NEXT >>
\]

This reads “for \( I = 1 \) to \( N \) do whatever the dots \ldots\ tell you to do. On each go around the loop counter \( I \) will be incremented by 1 until it reaches \( N \) then you leave the subroutine.

It might be helpful to see how one can put together a program with these pieces. Below I have a sample Modified Euler routine for solving first order ordinary differential equations.

**MEULER**

\[
<< \ "X =?\" \ " \ INPUT \ OBJ→
\ 'X’ \ STO
\ "Y =?\" \ " \ INPUT \ OBJ→
\]
‘Y’ STO
‘INCREMENT=” ” INPUT OBJ→
‘H’ STO
‘ITERATIONS=” ” INPUT OBJ→
‘N’ STO
{#0d, #0d} PVIEW DRAK
HOP
{X Y H N U V S T P} PURGE >>

HOP

<< 1 N FOR I
‘X’ →NUM ‘U’ STO
‘Y’ →NUM ‘V’ STO
‘SLOPE’ →NUM ‘S’ STO
‘X + H’ →NUM ‘U’ STO
‘Y + S * H’ →NUM ‘V’ STO
‘SLOPE’ →NUM ‘T’ STO
‘X + H’ →NUM ‘X’ STO
‘Y + (S + T) * H/2’ →NUM ‘Y’ STO
X Y R→C ‘P’ STO
P PIXON
NEXT
X
Y >>

SLOPE

‘g(U, V)’

This program is designed to calculate a numerical approximation to a differential equation

\[
\frac{dy}{dx} = g(x, y).
\]

The values you should enter for X, Y, H, and N are the initial values for x and y, the step size H for the modified Euler method, and N the number of steps you want the program to make. The first eight lines of the program just initialize these variables.

Next comes the graphics command

{#0d, #0d} PVIEW DRAK
This switches you into graphics mode (you are looking at something called PICT in memory). I believe the \{#0d, #0d\} normalizes the upper left pixel position on the screen to be (0,0). PVIEW requires an argument in this spot or it won’t work. I originally entered this as just \{#0, #0\} but the calculator modified my entry with the lower case d as shown. I don’t have a clue why, nor do I know how to enter a lowercase d from the keyboard. Incidentally, normalizing the (0,0) point for the pixel representation of the screen does not have any effect on the range of x and y variables in the graphics region. See the last paragraph of this note for one way to change these plot ranges. The last command on this line, DRAX, draws the x and y axis on the graphics screen.

The next line starts a subroutine called HOP that will run for \(N\) iterations (it does the modified Euler hops). Incidentally, you might think that you could just nest subroutines \(< < \ldots <\ldots\). However, it appears that the default behavior of a program encountering such a nested subroutine is to place the internal subroutine on the stack instead of running it. Named programs without quotes are run by default. I may have got this wrong since this seems very strange behavior but I do recall one of my many experiments going awry because a subroutine ended up on the stack instead of running.

In this subroutine the first thing that happens is that \(X\) and \(Y\) are renamed \(U\) and \(V\). Subsequent calculations will be done with \(U\) and \(V\) so that the initial values of \(X\) and \(Y\) are not prematurely disturbed. The next thing that happens is that the variable named SLOPE is evaluated and stored in a variable named \(S\). SLOPE is a separate program that should return the value \(g(U, V)\). For example if you are solving

\[
\frac{dy}{dx} = x^2 \sin(y)
\]

then SLOPE should be a program with the single line

‘\(U * U * \text{SIN}(V)\)’

The single quotes should appear in the program and \(U\) and \(V\) should be there instead of \(X\) and \(Y\) since as I mentioned above all the intermediate calculations are done with \(U\) and \(V\) until the final update of \(X\) and \(Y\). A simple way to create such a program is to type single quotes ‘\ ‘’, then put \(U * U * \text{SIN}(V)\) between the single quotes, then press the ENTER key to put the result in the first stack position. Then type ‘SLOPE’ and press the STO key. The entry in the first stack position will be stored as a variable named SLOPE.

Once SLOPE is evaluated at the initial value of \(U\) and \(V\) and the result is stored in the variable \(S\), the program makes a “fake Euler hop” to a new point \(U = U + H\) and \(V = V + S * H\). Then SLOPE is evaluated again (at the new \(U\) and \(V\) values) and the result is stored in the variable \(T\). Finally the average slope
\((S + T)/2\) is used to make modified Euler hop for \(X\) and \(Y\).

The program then hits the line

\[
X \ Y \ R \rightarrow C \ 'P' \ STO
\]

This converts the pair of real numbers \(X\) and \(Y\) into a complex number \((X, Y)\) and stores the result in a variable named \(P\). The next line

\[
P \ \text{PIXON}
\]

turns a pixel on in the display at location \(P\). Don’t ask me why but it does not seem to work if you use the array of real numbers \([X, Y]\) instead of the complex number \((X, Y)\). I also tried writing

\[
(X, Y) \ \text{PIXON}
\]

but this did not seem to work either.

The program then hits NEXT and repeats these steps \(N\) times. Before the program exits the subroutine it displays the final values of both \(X\) and \(Y\). \(X\) will show up in the second stack position and \(Y\) will show up in the first stack position. When you run the program it will spend some time drawing the graph of the solution. Once it finishes drawing it will return you to the stack display where you can read the final values of \(X\) and \(Y\). If you wish to view the graph again press the orange control key then the key labeled graph. To return to the stack display press ON.

The very last line purges all the global variables that have been created. If you don’t do this a long string of global variables will clog your VAR display.

One final word about the graphics display. You can control the location of the lower left corner of the display area \((x_{\text{min}}, y_{\text{min}})\) and the upper right corner of the display area \((x_{\text{max}}, y_{\text{max}})\) by editing the variable PPAR which should appear on your variables list. These two pairs are the first two in the listing of PPAR. If you want the lower left corner of the display to be \((0,0)\) and the upper right corner of the display to be \((10,50)\) then the variable PPAR should begin with

\[
\{(0, 0) (10, 50) \ldots
\]