Ti nspire cx ii cas programming language

TI Dispersion Programming Note that this page was written in 2009 and most of the content may be out of the way. For now, it is recommended that you visit naspiforums if you have questions. Or see the TI Sapsloware manual for complete documents. Update: Now you can go to the Spispair Home page ti-Spischis is a unique calculator in the basic series. The calculator's programming system is noted to be similar to 68k hardware, but because the naspika internal core is different, it needs a portion on its own. Unlike all older TI versions, NASPK does not have a home screen or graph screen. Instead, the calculator run on a document that has five applications: calculator, graph and geometry, spreadsheet, note and data. The programming calculator works under the application. Although naspiform and screen resolution are well used, programming is much less than the older version. This article is to give you a brief overview of programming in the naspi and how many older versions are preferable if you want to program. The Pesser: Introduction Texas Antilles Nasipri is an upgraded version of all other TI models. It uses a lot of formatting like 89 family but does not have AS. When you first observe the calculator, you will feel your pad. It is very different from any other calculator. It has a much weirdlayout compared to the other TI graphicing calculator. First all the letters get their buttons which make it a little easier to type. In addition, it has ctrl key instead of the other key. Enter the clickor key, menu key, catalog, and even the key of the hat. If you observe it long enough, it seems to match the keyboard. This is because the nasili is made like a computer in a way. This information uses documents to store. These documents are information about any work you do, not just programs. You can choose to save it or load different documents like a computer. In addition, Sspeser uses mouse type systems. This arrow replaces the keys with a round button. It acts as arrows, and as a way to move mouse-like marks across the screen. The nassia is the transfer of a calculator to the computer, yet is kept behind enough to maintain the calculator status. The other thing you will feel is screen resolution. very good! The screen is much better than any other calculator in history. The screen shows very well curved, it graphs cleanly, it uses different size letters and other characters (much like computer font style), and it's larger. Also, the screen has many different colors of the color of the color. Using non-natuanis, the calculator can actually shadow the region rather than draw the lines it represents, and it darkens the overlap swells. The screen is very well-made. The last most remarkable thing is that you can lose on where every function is. The calculator has a very different key layout, and its Also different You may have trouble finding where some functions are. This article does not discuss the way to find different functions, but rather discussed the programming attribute that is the main topic of this site. To find different functions, see the catalog (book button) or check your CD manually. It has a load of information you can use. Programming for NASPIIs is very similar to the 68series. However, Spersper is not just part of it. One is, Naspicanhas have no graph screen or even a home screen. Everything is formed by documents, issues, and pages. You can do a document, issues, and pages within this document. Although everything inside the document is connected in some way, the calculator does not seem to have the ability to access other pages or issues without manual control. The calculator does not have any graphing orders because of this delimity. He lost commands like PTOn (and line). Anything under the draw category has been deleted. The draw command was the basis of high graphics, so that means Naspi cannot create a game with any type of graphics, although its resolution is much better than the old models. Further, the calculator has no home screen. Instead it uses a very strict line through line layout. The calculator is very systematic about an operation or command. Unless the calculator completes the assigned command or operation, it will not allow the user to interfere. The new information is automatically in a new line. It looks like other calculators are completing the task without any constraints. However, it becomes a big problem when it is related to programs. It works the same with the nassia programs; it is not allowed to interfere with the user. This means the calculator will not accept user input until the program is finished. This defeats the purpose of the programs. What is programming using if you can't input anything? It happens that the program accepts input before the program is executed, but not during. So, to create a program with the same impact of programs in 68k or 83 family basics, you'll have to create multiple programs that must be accessed individually, which gets unbelievably cumbersome. So, the calculator in programming is awful. It cannot create graphics or advanced layouts and will not accept input. So what is a program used on the nassin. True, if you want to program, don't get the nassin. In simple words, there is no real way to work around these obstacles. However, regular Spacle comes with an 84+ keypad. It has all the capabilities of regular TI 84+. The thing is, although the calculator has the capabilities to use extreme resolution, the 84+ speed does not use it. So it's still not that good. So what if I already have a sin? do not worry. If it comes with an 84+ speed, go ahead and use it to program. In any case, Naspi is definitely better in mathematics. It has achieved many useful things by sacrificing programming capabilities. Tools. For example, it tests primes, triangular functions such as secant and co-outant, a new improved digital solver, point plotr, geometry grapher, spreadsheet, and more. Even when it uses templates, instead of writing math equations ((3+4)/(5(6+3)^4, the calculator can write it as $\left({3+4 \over 5(6+3)} \right) ^4$ instead. NASPIR can also catch more than 1E100 numbers. So, it's better in mathematics (which can be a real big help in math tests!) But not a good programming apparets. This article will further discuss the topic of programming on a naspile calculator that describes how to create programs, all commands, input methods, ways around major obstacles and finally a game example. To create a program or action making program, make sure you are in the calculator application. Press the menu and go to 8. Then, press 1 and 1 again. A window will name the program. Insert in the name, select type (program or function), and press OK. Now, you're in the program editor and you can now create your own program. Programs and functions differ. Depending on what you're doing with the program, you want to select one of them. Program function return a result? No yes can be used in an expression? No yes, can you run in applications other than calculators? No yes can use all commands? Yes, no one can use multiple variable types? Yes, can you access global variables? Yes, yes, we can modify and create global variables? Yes, can't call a subprogram? Yes, can't call a function? Yes one program is what other calculators typically use. Functions can be used in line in expression, such as 5+func(7)-2. However, the functions do not support most commands on this page. Format TI Nasper format is a few more organized than 83 series. The program editor has a lot of the same format except pot for each group of commands. For example, it does not pot if... Block again, while... End while blocks, and others. Explain the test(a,b)= Prgm if a=1 then ... Inzip 1 ... c:=5 Other if a=2 then ... Inzip 2 ... While b>0 ..... b:=b+2c ..... Inzip b ... End while others if a=3 then ... The Endif Endif EndPmrm dispersions the 68k variables very much similar lying on the naspper function. There may be a string of variable characters. An undefined variable is atalxed. For example, if the variable height was unfixed, or still had little storage, the calculator would look like a variable height. Defined variables are bold, so if height is defined, it will appear as height. Commands are neither burnt or italysed and you cannot store information in them. To execute the implementation program, go to the Calculator application. You must then type the entire program name. Then put the parentheses with the values that are defined within them. The recipe for the above will look as follows: You can also select the program from the var button. On the commands of the nassin There are most commands Or 68k command seres. However, Sphysyellow lacked most of the scriptive commands, and some variable commands are limited. Some commands need a computer algebra system or AS, so if you don't have AS in your device, many commands will not work. There are a few commands for special use in programming. These commands help direct how the program works so that it can complete different tasks. The orders are divided into categories based on their overall function. Define variables These commands are related to variables. The orders listed here will affect the way the program treats the variables. Local local is an order that tells the calculator how to identify the variable that follows. Tells the local program to make the variable a temporary variable that works only within the program. So, when you local is a variable, that variable is only used within the program, and if the same variable is used somewhere other than that, you will know that it has not been affected. Here's the recipe. The name can be any variable name or type. This command is useful in the sense that it will not mess with your previous use of the variable. If you want to use a variable name, but it is used in other programs and you don't want your program to mess it up, use local. If there is a case where you want the variable to be changed globally, then do not use local and the calculator will treat it as any other number. Define command Explanation will allow you to use subroutines within the program. This command creates a new program or task within the program so that the subroutine can be reused. To make it work, you must either be P.G.R.M. End-PG-RM or Fanc... End funk for her to work. Test2(x,y) Pmrm Disp x,y EndPmrm Define delete variable This command will delete a variable by making it undefined. This command will not set the same number as 0, but instead it changes the variable to become unusable until later defined with the storage command. It would be good to use this command when cleaning the program. If you do not want your variables to have values after the program is finished, use the dealwise to reset the variables. Fanc... End Funk and PGM... End-PGM can be used to make subroutine sits using them with clarity. Fanc... End Funk will create a function while The P.G.R.M. ... End-PGM will create a program. Use programs after they are defined to create them, which can be explained later in the main program. Test 2(x,y) Explain the Pmrm Disp x,y EndPrm control These are the main commands that will be used. Control Command controls the flow of the program, which influences which commands group to enable or how often to start a group of commands. If it is exactly the same if the condition used in other basic languages. You placed a condition after if, and if the condition is true, the next line will be implemented. If it's wrong It will be left out. Here's the recipe. If height >0 height*width→ area it is very useful in programming, if not necessary. If command makes the game It detects limitations, anaesthesia, and special circumstances, which can affect the reality of any program. what if... And... ENDIf IN BASK, if it is equal to: A condition if goes after. If the condition is true, everything will be implemented by the end. If this is wrong, the calculator will start with end-f and put these commands into practice. Here's an example of this. If h=3 then ... 3+3→a ... a+h→c ... c/h+a→ h Endif disp h In this example if h is 3, the number shown will be 9. If h is not three, it will show any number that is not three. Then, this conditional command is greatly valued in programming. This is important in programming! It also determines boundaries and special examples, but it open up a new field of programming. what if... And... Other... Endif we probably don't have to tell you that it's the same in the basic. It has one such if conditional after that and then. If the condition is true, then everything after that will be implemented but after the second everything will be left out. If the condition was invalid, then everything is then left and everything is left after the other has done. Here's an example of this. If a > b then ... Yes other DSP... No Endif in this code if someone is greater than b, the program will say yes, otherwise it will not say no. This command is very useful. It is an effective way to test something, and then the program does some work in terms of its decline. This command control is a great way to move your program, and it is very recommended to use it. Second f... So this is a special form of if... And... Other... End command. This command allows you to test multiple different options instead of just one. This command should come after a if... Then, and you can have a lot and if so. First, if the first condition is correct, it will know. If so, everything that follows is executed and the program goes to Andf if it is confronted. If this is wrong, the status after the next second f is tested. This does so until all other conditions are followed or one is correct, and the following commands are implemented. If a=1 then ... The inzip is a one... 4→c Other if a=2 then ... The inzip is a two... 2→c Other if a=3 then ... The inzip a is three... 8→c Endif This code tests the value of three times. First it turns out if a=1. If this happens, text a is one will appear and c will be four. Then it goes ahead after the endof. If a is not one, the next condition a=2 is tested. If a is two, text a is twn is displayed and c gets 2. This is how another f works. The other if is a lot like... And... Other... Andf. These two codes are the same. If a>6 then ... 2h+3→c ... a+2-h→ d ... Next move? Other... D.S.P. You have to give up... If a>6 then stop ending... 2h+3→c ... a+2-h→ d ... Next move? Second ≤6 so... D.S.P. You have to give up... Stop Ending for... End for this calculator is a loup. This command, when used as variable, start, stop, start optional addition, store it And enforce every command until it is finally reached, then the program will go back to the fore, and add optional additions to the variable and do so until it has reached the number stop. Then, it will finish the loup and start following the commands after end-lines. Optional addition is an optional input, and the program will assume that it is 1 unless otherwise stated. Here is an example of a for-help process. For one, 1,11,2... The 1, 9, 25, 49, 81 and 121 displays will be displayed in the display a2 and this code. This is because it starts as 1, then 3, then as 5, and so on. While... End while this command is another type of a loup. This is where the loup needs to be conditional. It does the loup as long as the condition is correct, and the loup ends when the condition goes wrong. To execute the lup, the condition must be correct initially. In addition, make sure that the changes within the state change otherwise the lotude will become unlimited. 0→x while x<5 ... disp= x= ... x+1→x= endwhile= disp= x= this= loop= will= display= 0,= 1,= 2,= 3,= 4,= and= then= 5.= the= loop= ends= when= x= becomes= 5,= and= it= displays= the= numbers= from= 0= to= 4= because= the= starting= number= is= 0.= while= loops= are= essential= to= game= creating= or= animation.= loop... endloop= this= command= creates= an= infinite= loop.= this= type= of= loop= requires= no= arguments= and= will= not= stop= unless= you= insert= a= stop,= exit,= or= goto.= to= exit= the= loop,= it= is= best= to= create= conditionals= within= the= loop= to= signify= when= to= jump= out= of= the= loop.= 0→x= loop= ... disp= x= ... x+1→x= ... if= x=>5 ... Exit End Disp x This code shows the same thing as when... End by the top of the top. The only real difference is that with the ... End-of-the-day, you can create a condition inside the ... Checks the end while conditioned at the end only. Try... Other... This command allows you to literally try a group of commands, and if they make a mistake, others have started later. If an error occurs in the effort part, the calculator is able to get off the error and pick up on the other. Try Delvar b... 6+b→c //Note that b is undefined other... 6+7→c Displays end-of-the-display sp c program 13. The program needs to try → 6+b, but since b is undefined, the program errors. However since it is in one Trit block, it leaves it to another and

6+7→ c which works. So, it shows c that is 13. This command clears the current error and sets the error system variable to zero. Pass-the-pass-ris ser will pass an error on the next level of an attempt... End block. Transfer orders direct what a program does without the need for input. These commands only follow when they are in operation and tell the program specifically what to do. Return this command will exit the subroutine and directly progress to the parent program. The Cycle Cycle Command program works like an initial end in the luup. When confronted, it's immediately Shoot back at the beginning. When the loup is over it does not go on the cycle but it goes to the right end. Local a 0→ a 5<10... disp= a= ... a+1→a= ... if= a=>... Cycle... a+1→a</10> </5> </5> This code displays 0, 2, 4, 6, 7, 8 and 9. What happens is a+1→ twice as a+2→ a. However, when one is greater than five, the other is not a+1→ because the cycle restarts the lobe. Exit This command will immediately exit the current lotyou are in. So, it prevents the loup and starts where the end is. This command is used to avoid the ... End-of-the-lot, but it can be used with... For the end and while... While ending. No arguments are required for this. 0→x Inzip X ... x+1→x ... If x>5 ... Exit End Inzip x Lbl is labeled in a program. A label is like a flag or signer in a program that can be used as a location through which a program can be used by using goto. The label has a name after them, such as Lbl here, when the program faces the goto name, it starts searching for the label with this specified name. When this happens, the program resumes on this occasion. Go to Lbl this will create a Goto in the program that will send the program out to find similar labels. It can be used to jump or use entire parts of the program as a loup. Local a randInt(1,2)→a if a=1 Goto 1 Disp Head Stop Lbl 1 Disp Tails This is a simple coin program that Goto uses. First, a random number is stored in local variable a. If a is 1, then it is tail. The program competes with Goto 1 and thus goes to Lbl 1. From there, this program ends. If there is one or two, goto 1 is not confronted, it shows the head and stops. Stop this command program. It does everything. When the program faces this command, it will completely end the program wherever it is. 3:=a Stop Desp How did you get here? This saves code 3 to a and expires. The code will not give any output at all. This is useful when you arrive at a location where you can use user information and end commands (other than ON) so that you type a statement to end the program immediately. Then, you can then put the goods in if the user did not want it to stop. The stop is also junk at the end of the program. I/O Displays the command on the screen whichever comes after it. This is what you see when you run the program; it's what the calculator produces as a result of internal commands and calculations. Simply put what you want to show after DSP. This will literally show text if you use the excerpts. If there are no excerpts, the calculator will try to resolve the expression you offer. This code will show the answer to ab+a and then show the hello after that. This command is required unless you tell the user something that is intended for the program. So use this command, because it is the only way to communicate the message to the user. Application Syncax: Application Compact String, Var[, DispGlog [, StatusVar] or Application Compact String, Funk(arg1, ... [argn], The Display [, StatusVar] request receives the number from the user. It pops the dialog box with your programised selection message. For example A small program piece: recprorug()= Prgm Enter a number: Number If number=1 then (code) Endif. . Request Str Syncax: RequestStr Compact String, var [, DispGlog [,Status-wise] is almost the same thing as the request, but requests a string. Specify your name()= PGram Recoast ster What is your name?, Name Desp Hello, Name Desp Your name, Fade(Name), characters long. At the end, the gram text pops up with the Sync: Text Logy string [, DispGlog] message and the Dialog box with the OK button. randprgm()= Prgm for i,1,5 strenfo:=Random number & string(rand(i)) Text strainfo Next end pergram mode are those that change the overall production and computing system of calculators. This includes changing things from degrees to reedin, and from rectorto polar. In normal circumstances, these commands may not be used unless you are making a math program. Getting input is rather difficult and extremely limited to get user input on the naspper. The main problem is that the only I/O command is DSP. Unlike the TI-83 series or 68k, naspi does not have input, output or gain key. The lack of these three commands makes programming quite weak on the calculator, although it has dozens more capabilities. But what TI has done is they expect the user to input variables before the program is implemented, and that is usually the only way the naspi programs will get input. The problem is the user does not know what he is going to put. With updates from The Spair OS, the request and request ster are included in the command list, which enables data input while the program is implemented. Input is still limited, because you can't create a menu like the TI-83/84 series. In internal orders, they have I/O message boxes (wizards) with multiple input for scroll lists and commands; unfortunately, TI has not enabled programming of such dialogue boxes. There are three ways to receive input. The first way to store the user's desired values in all variables before the program is to be processed. For example, say that a program uses variable width and height. The user can save whatever he wants to be width or height using the suntax value → height. If there is already something in the variable, then this value will be → height. Then, the user can execute the program. The second method is within the cause of the parenthesis that is finished during implementation. In the Program Editor, the programmer can go to the top line where the defined program (). Then, you put the variables you want into a separate parenthesis through the kama. So, still using width and height, the line above will look like this: Specify the program (width, height). Then, the user typethe name of the program with the parenthesis, but instead of leaving the parenthesis blank, the user will put the values he wants to assign width and height to the parenthesis separated by the comma. The last method is to get input during an important program. Unfortunately, the only way to receive input mid execution is to completely shut down the program and let the user enable another. (or the same) with new values. Of course, it's not all user friendly. It can also be a cause of a bet to explain a program again every time. Programming Is Really The Only Real Application For Programs Is The Use of Math and Probability Programs. Although the idea of sports is a concept, it would be invain to try to create something like a phang or another simple game. Even making an guessing game is almost impossible. Nevertheless, programming applies to helping you with your math or giving you an idea of the possibility. Math programs Making math programs can be quite useful. These programs are those that ask for input and start solving. It can be easy to program as most math programs initially only need one input. One such program will be a quadributequation solver. To create a math program, you must first determine what you are solving for and what variables are being given. For our quadrax equation solver, we want two roots, determinals and ras when we insert a, b and c when ax2+bx+c=0. The user must input a, b, and c at the beginning of the program. Since it is the only input, it can be done easily. Define the program and put the variables inside the parenthesis. Define The Co-a,b,c) Pmrm EndP.rm The next step is to plug your formulas. We want the calculator to solve for roots with a special condition: for two roots, it must reveal both roots and two solutions, one for root, answer and one solution, and to reveal a single solution when both roots are complex. We should first find the determinalthat will tell us how many solutions will result. Explain the coadathtic (a,b,c) Pmrm b2-4a*c→d EndPmrm Now, we can add our conditionals with if... Endoff blocks. Define the coadatic (a,b,c) Pmrm b2-4a*c→d if d>0 then ... DSP Two Solutions... The $b+\sqrt{d} \over 2a$ ... The $b-\sqrt{d} \over 2a$ Endif if d=0 then ... An zip solution ... The $b \over 2a$ Endif if d<0 is not a solution. Probability programS This is probably the most imagined type of programs you can create on the bottom. This will be the type of program where you experience the possibility of occurrence under certain conditions. It will be like the spinner turns 10 times or flips the soin 25 times. It's easy to create this program because you just need a random command and a laptop. For this program, we are going to say that we need to rotate a spinner 100 times where there is an ideological potential of 0.5, yellow 0.25 and red 0.25 in the blue. First, explain your program. The user does not need to define a variable, so we will just close the parenthesis on the first line. Explain the spinner () PGRAM EndPgram now, we want to define our variables within the program, so we need to localise them. We want our variables to be named blue, red and yellow. Also, their origin. Define the spinner () PGM local blue blue local red local 0→ اسپنر کی وضاحت کریں ہوں گے۔ پر ختم ہو کر 100 شروع ہو ایک سے اس لیے ہم 100 ٹرائل چاہتے ہیں، اس لیے ہم شروع کرنے کی ضرورت ہے۔ چونکہ ہمیں لوپ کے لئے پیگروم اگلا, مقامی نیلا مقامی سرخ سرخ مقامی پیلا 0→ نیلا 0→ سرخ 0→ پیلا ایک,1,100 کے لئے ... اینڈ پیچی آر ایم ناب کے لئے، رنگ کو تلاش PGRM () نیلا 0→ سرخ 0→ پیلا اینڈپیگروم اگلا, ہمیں randInt () اسپنر کی وضاحت کریں کرنے کے لئے PGRM () استعمال کریں چونکہ ہم چوتھے سے نمٹ رہے ہیں، اس لیے نمبر 1 سے 4 استعمال کریں اسپنر کی وضاحت کریں کے لئے 1,100,پیلا ایک→0 سرخ→0 نیلا→0 پیلا مقامی نیلا مقامی سرخ سرخ مقامی ... randInt(1,4)→spin EndPgrm اگر متغیر تاکہ ہم کی ضرورت سیٹ اپ کرنے ہر مشروط ہمیں, کے لئے اگلا spin 1 اسپنر کی وضاحت کریں ہیں۔ استعمال کر سکتے دیگر ایف احکامات کے لیے ہم اس سرخ اضافہ, اور اگر یہ 4 ہے, پیلا اضافہ, اگر یہ 3 ہے, یا 2 ہو, نیلا ایک اوپر جاتا ہے PGRM () کے لئے 1,100,پیلا ایک→0 سرخ→0 نیلا→0 پیلا مقامی نیلا مقامی سرخ سرخ مقامی ... randInt(1,4)→spin ... اگر spپن <3 then= …… blue+1→blue= … elseif= spin=3 then= …… yellow+1→yellow= … elseif= spin=4 then= …… red+1→red= … endif= endfor= endpgrm= now,= the= last= step= is= outputting= the= results.= simply= create= a= list= of= displays= to= show= the= number= of= spins.= define= spinner()= pgrm= local= blue= local= red= local= yellow= 0→blue= 0→red= 0→yellow= for= a,1,100= … randint(1,4)→spin= … if=></3> <3 then= …… blue+1→blue= … elseif= spin=3 then= …… yellow+1→yellow= … elseif= spin=4 then= …… red+1→red= … endif= endfor= disp= yellow= disp= yellow= disp= blue= disp= blue= disp= red= disp= red= endpgrm= game= programs= the= game= program= is= very= hard= to= accomplish= in= nspire= basic.= these= types= of= programs= will= usually= take= more= than= one= program= to= accomplish= in= order= to= accept= more= user= input.= however,= a= game= such= as= guess= my= number= can= be= done= in= one= program.= the= general= strategy= is= to= create= a= starter= program= that= sets= up= the= game.= then,= every= time= user= input= is= required,= separate= it= into= another= program= and= leave= a= display= telling= what= the= next= program= is.= for= one= program,= initially= start= a= global= variable= with= a= strange= name= so= the= program= can= recognize= if= you= are= starting= a= new= game.= here= is= a= game= where= you= have= to= guess= a= number= between= 1= and= 100= in= seven= tries.= first ,= you= need= to= create= variable= gmn= as= 0.= this= variable= will= be= our= deciding= number= that= finds= what= try= you= are= on= and= whether= a= new= game= starts= or= not.= create= the= program.= variable= a= will= be= the= user= guess.= define= guess(a)= pgrm= endpgrm= define= what= happens= if= gmn= is= 0.= the= program= stores= a= random= number= between= 1= and= 100= into= b.= then,= 1= is= stored= into= gmn= to= represent= guess= 1.= define= guess(a)= pgrm= if= gmn=0 then= … disp= guess= a= number= between= 1= and= 100= … disp= type= guess(=)= with= the= guess= in= the= parentheses= … randint(1,100)→b= … 1→gmn= … stop= endif= endpgrm= this= next= part= says= that= if= your= guess,= a,= is= larger= b= to= display= guess= lower.= if= it= is= smaller,= guess= larger.= if= a= is= equal= to= b,= then= it= displays= you= win!= and= the= gmn= resets= to= 0.= define= guess(a)= pgrm= if= gmn=0 then= … disp= guess= a= number= between= 1= and= 100= … disp= type= guess(=)= with= the= guess= in= the= parentheses= … randint(1,100)→b= … 1→gmn= … stop= endif= if= a=>b انزپ زیریں دوسری ... تو <b Then … Disp Higher ElseIf a=b Then … Disp You Win! then= … disp= higher= elseif= a=b then= … disp= you= win!=></b Then … Disp Higher ElseIf a=b Then … Disp You Win! > اگر a</3> a</3> ... End end end is that finally, the program will check if you live. If the GMN variable (our turn-per- 7) is greater than 7, the program is finished and the gmn is reset to 0. Explain the estimate (a) Pgram if the gmn=0 then ... Guess a number between DSP1 and 100... Type the estimate with the estimate in the insal parenthesis... randInt(1,100)→ b ... 1→gmn ... If ab is > Close... The second under the inzip if a<b then= ... disp= higher= elseif= a=b then= ... disp= you= win!= ... 0→gmn= ... stop= endif= gmn+1→gmn= if= gmn=>7 So... Desp you defeat. ... The answer was the inzip... Inzip b ... 0→gmn Endif EndPmrm A game example this game called Fighter X is an examination against the calculator that will give you how well you guess. This is a relatively simple program where all users estimate numbers from 1 to 3. If you guess what number the calculator was thinking about, the calculator lost HP. Otherwise, you lose HP. Try to understand the code. Define Program 1() Pmrm 100→you 100→calc Disp Are you ready? To start the inzip (to start) type gs with your estimate in the EndPgram program 2 explain gs (g) Pgram randInt(1,3)→bc if g=gt... calc-randInt(5,13)→ calc... you-randInt(2,7)→You Endif Discuss You Discuss You Disp Calc If you type 0 and calc>0 disp gs(and re-estimate endPgram brakes a calculation on-break naspversion <1 then= ... disp= you= lose= ... delvar= you=elseif=></1> 83/84 A little more than the series<1 then= ... disp= you= win!= ... delvar= calc= endif= if= you=>different. For Spsples, you must press and press the on button and wait until it confirms. Then, you can break the calculation. It is useful in the case that an overly complex function is being performed and you want the calculator to stop thinking so that you can do something more, without reset the calculator or remove the batteries. Click here to visit the Forum TI Spesser Forum. If you have questions about Spspyr programming or calculator yourself, please ask on forums, and someone will help you as soon as you can. Further research You can explore a lot about Naspi and its commands through 68k part of this site as they are quite similar. Page 2 TI-Bisc Developer Home: Welcome to 68k TIBsk Developer (TI| BD), TI-Basic Information Store! If you are a visitor for the first time, please check the welcome pack to get speed when you use the site. We inspire you to become a member and join the community, and often come back and see what changes have come. And above all, enjoy your stay! Search site This site has a wide range of 68k basic content available, so we recommend using the search engine or referring to the site map. If you're not getting what you're looking for, leave a post on forums and someone will help you. Page 3 68k</1> Command Index is a superscript next to some commands that indicates compatibility: 2.xx indicates that the command THE AMS 2.xx or more required on TI-89, TI-92 or TI-92+ (or any 3.00+ version, on TI-89 on the titinium V200). 3.xx indicates that the command needs ti-89 titanium or V200 calculator, as well as AMS 3.xx or more. Flash indicates that the command needs calculator with flash ROM (that is, it will not work on a TI-92). Page 4! Command Command Summary takes a number of factors. Command Syntax Number! The menu location cannot be found on any menu except the command catalog. Calculator compatibility This command works on all calculators. Token size 1 byte! Operator takes a number factor: positive, for the whole number, n! Is interpreted as n*(n-1)*(n-2)*...*3*2*1. As a special case, 0! It is interpreted as 1. Elementin the computer has a special meaning: n! There are number of ways you can order n objects. For example, 3!=6 because there are 6 ways to order 3 items: A B C A C B B A C B C A C A B C B B A As can be expected, the factors become very large very soon. Calculator can count the correct number result for factors up to just 299!, and an approximate result for factors up to 449!. Beyond that, the numbers included in it are ∞ to the expression (unlimited). While there are some formulas to explain the factors for the non-correct number values, the calculator does not use them. It will leave an expression like (1/2)! Uneasy. However, the factor of number less than -1 or equal will be undeified. :5! 120 :299! 1020191707388... (600 more numbers) :449! 3.85193e997 :(-2)! Undef-ComputerCommand-Path 5 Frequently Asked Questions (FAQ) This FAQ is an attempt to answer common TI-bask related questions that people ask. Many questions are related to each other, so it is recommended that you read through the entire list. If you have a question that is not mentioned in the list, please post them in the forum or leave a comment at the bottom of the page. Question: Is it easy to learn TI-Bisc? A: Yes! TIBasic has the majority of standard features and functionality that you find in other basic programming language variables (i.e. things like user input and variables are very similar), so tibasic should not be a problem if you can learn those languages. If TIBasic is your first programming exhibition, it will need some work to learn, but it's definitely worth it because TI Basic is a fun language to use. Question: How do I learn TI Bisc? A) The best way to learn TI-bisc is to download a copy of the manual and start creating small, sample programs to try out different TI-bisc commands. After feeling comfortable with commands, you can start keeping them together to create larger programs. Then, you should move towards learning more modern design concepts and techniques that are part of the TI Bisc. Question: Where can I get information about TIBsk? A: The wiki you are currently on has the largest collection of TI basic information available, including commands, design concepts, techniques and experience. The Resources page also has a comprehensive list of TI basic tutorials you can find On the Internet. Also, you can download all these tutorials (and more) on the Downloads page. Question: Do you have a tutorial on [the subject]? A: The best way to find out is to use the search box. If you don't find what you're looking for, leave a comment on the forum and one of us will try to help you. We will not guarantee that you will find everything you are looking for on this wiki, because it is a constant task going on and there are only so many topics to cover. If you want to make a proposal for a new tutorial, you can add it to the Doe list from Wiki. Question: Where did the TI-Bisc name come from? A: Back when language was growing in popularity and use, people wanted a simple name to be referenced that was easy to remember and tell you what it was. Because it is the built-in programming language of the TI graphicing calculator, and it is a variable of the basic (more or less), TIBasic is what he called. You should note that the name is unofficial, because TI has never actually named it (for example, try searching for TI-basics in the calculator menu; you will not find it). Question: I've seen ti-besk magic with all the upper case (TI-bisc) and mixed case (TI-bisc), but what's the right way to spell it? A) True, there is no correct way to spell it. This is just a personal preference. However, on this wiki you will probably see that we spell tibesk with a mixed case. The main reason for this decision is that reading is easy (not all the hat is too reader friendly). Question: Which calculators support TIBSK? A: All TI-bisc support of TI graphics calculator sarebuilt in. Of course, the calculator is everyone's own TI-bask variable (see next question). Question: What is the difference between Z80 TI-Bisc and 68K TI-Bisc? A: Simply put, a whole lot. Z80 TI-Basic slacks all the things that are in 68K TI-Basics, including nondirectional, local variables and functions, high picture breaks, text in metrics, and so on. It is also a shame because these things are extremely useful and make TI Basic sire much more rich than a language. QUESTION: Is there a place where I can interact with other TIBsk programmers? A) Although there is a forum available to discuss and help ti-besk on this wiki, the best TI-Bisc forum in terms of user activity is United TI, with a 68k TI-Bisc section. Actually, the majority of members of this wiki are active members in united TI, so you'll probably see us hanging there. Game Q: Where can I find TI basic games and programs to download? A: On our resources page, you'll find several links to sites related to general TI. A great site for going to sports and ticalc.org, which has the largest archive of ti-besk games and programs of any site on the Internet. You can also browse our own program archives, which can be found here. Question: What is an emulator? A: Emulator you on your computer Allows the virtual shape of the calculator to run, which is It's easy when you want to make quick changes to programs, or make a debuging or better. There are several emulators available for your use, so you should experience just to see who you prefer. Question: I downloaded an emulator for my calculator, but it will not work because it says it needs a ROM image. what is that? A: Rom image is just an example of your calculator, which tells the emulator that you own your calculator. It is primarily used as a protection because only one person is considered to use a one ROM image. To download a ROM image to your computer, you only link your calculator to your computer, and then the emulator must be able to download a ROM image from it. Question: I have a great idea for a game, but I don't know how to program. Can you program it for me? A) When we want to help you program your game, we're each one of our own projects that we're working on and other real-world things (like schools and jobs) that occupy our time, so we're not able to program your game for you. At the same time, if you have a specific TI-Bisc programming question that you need help with, we'll be happy to help you. We're also doing a better program, though, are you programming it yourself (see next question). Question: Why do I need to make games? A: The important things you need to make the game are your TI calculator and calculator menu. Before you really implement a game, however, you should plan it. This includes come up with the idea for the game and working on many details of the game: graphics, game play, menu, and so on. Once you're assessed all these things, you just need to put them into action. Q: What is a good tutorial to make games? A: Unfortunately, there is really no comprehensive sports tutorial available. Instead, there are several small tutorials that each covers different aspects of sports. Also, this wiki covers a lot of techniques — see the special topics section, for example. QUESTION: Can I use any routine from this wiki in my game? A: Yes! In fact, we encourage it. All the routines on this site are designed to be as good and effective as you can, so that readers learn the best way to program. Question: Can I use the sparety from other sports in my game? A) The general consensus in the calculator programming community is that someone's graphics are ok in your game, as long as you'll be allowed to. However, if you don't plan to release your game to the community, but instead keep it to only you and your friends, then it doesn't really matter. Programming Q: How do I become graphics? A: 68k TI-Bisc has many graphics commands. Probably the most useful are *Pic commands (such as XorPic, of any size anywhere on the screen Can be used to display. However, other commands, such as lines, are useful. Question: Can I [work] in TIBesk? A: While it is possible to do almost anything in ti besk, whether it looks good and and There is a different matter at decent speed. If you have planned your program fully and improved it as much as possible, and it still took a minute to load your program and five seconds after each important press, it's a good indication that you should probably use assembly or C instead. At the same time you should always try to push the tibesk limits. Question: How do I convert a number to string and vice versa? A: The String () command can be used to convert any variable type to string. To change in the other direction, use the command rector. QUESTION: My program is very big. Is there a way to better manage/exempt code? A: First, your program will probably shrink in size after running for the first time, due to tokenization. If this is not enough for you, see the Improve page for more tricks. Question: Do TIBescs have any non-documented features (Easter eggs)? A: Unlike the TI-83 version of TI-Bisc, in fact everything we know about the 68k calculator's TI-Bisc is documented somewhere. However, there are some features that manuel does not emphasize, but TIBSk is quite useful to the programmer. An example is the alternate parameters of the Set Mode () command. Question: How do you disable the on-key? A: In pure TI-bisc, it is impossible. Using The Effort. Endblock, you can disable the ON key during text input, but the only way to disable it globally is with the Assembly program or viable code. Question: How do I hide the code for my TI Besc program? A: It's impossible — if someone can run your program, they can also see the code. You should not really try to hide the code too, but let others learn from it instead. Rectifying Q: My calculator cannot handle the effects with several variables(for example this element cannot x^2+y^2+2xy). How do I fix it? A: When you type xy next to each other, the calculator does not think of it as multiplication but as a different variable called xy. Write x*y instead and you should not have any problems. Question: I've entered a simple expression and trust the result, but the calculator gives something strange! What is (for example, d(x^2,x) gives 10 as a result? A: Make sure that all the variables that you are manipulate in the normal way (x, in example) are actually undefined. If not, delete them using a dealwise. Otherwise, their values will get substituted and you will get a strange answer. This can result in other surprising mistakes. Question: I think there are some routine mistakes on this wiki because they did not work for me. Can you please correct them? A) We have tried to ensure that all routines on this site work correctly and without problems. However, if you are 100% sure that you have entered the routine correctly in your calculator, please leave a comment on the page using the comment function at the bottom of the page. After that someone I will be able to correct it so that no one is in trouble. Question: I was playing a TI Bisc game and mine mine Suddenly stopped. When I turned it back, my memory was over. what happened? A) There was some kind of a breakdown in your game, and that caused the calculator to crash. This is usually due to assembly programs, as the majority of TI primary errors are in the grip of calculators. You don't have to worry too much about the TI-Bisc sc' connection because they do no real permanent damage to the calculator, but because changing all your programs after your RAM is cleaned is very disturbing, you should always save any important files in the archives. QUESTION: I found this error message when I tried to run my TI-Bisc program. What does that mean? A: Most error messages are largely self-defined, but if you are still confused, you should consult our list of error messages with more depth clarity. Question: Transition program, TI Connect, does not work for me. How do I fix it? A: TI connectivity issues can be a problem on both Mac and Windows platforms. Here are some of the listed solutions for Windows and Mac platforms. Mac: Install and reinstall TI connect Toi Connect Manager X. This activity can be done using a monitor. Start activity monitor in /applications/utility, then search for TI Connect Manager X in the process list (if you can't find it, just type T on your keyboard). Select it, then click the X icon in the top left corner of the activity monitor. Click Leave Force. You can then restart device manager, and it should detect your calculator. In some cases, disconnecting or closing your calculator will always cause ticonnect manager X to crash, and you will need to connect it to your computer every time. To simplify this process, there is a script that you can use to auto-create this process. Use a different USB port. Windows: Then install ing must fix most of the issues. Check to ensure that the plugs (at both sides) are firmly in the harbour. Assembly/CQ: How does TI-Bisc compare to assembly or C? A) It's very easy to learn and program in TIBsk, but it's slow because it's an interpreted language. TI-Bisc se is a number of good graphics commands, but still will be softry compared to assembly or C programs; in addition, the TI-Bisc programs are limited in control of the calculator. Question: Is it possible to convert TIBsk into assembly? A: No, it's not. There are currently no working programs available to convert TIBs into assembly (Note: I say that are working because people have tried to make TIBs into assembly converters, but no one has yet completed it), the only way you can convert the TIBSc program into assembly is by learning assembly and porting the program itself. You can also try to ask an assembly programmer to port it to you, but most people will not do so unless the program is too small. However, if you own If you want to speed up the program, you have the option to use Basic Builder 3.0, which packages your TIBasic program into the application. QUESTION: I want to use the Assembly With my TI BISC program, but I can't figure out how to use it. can you help me? A) Unfortunately, we can't really do much for you. What we recommend is to contact the author of the Assembly program and ask them for help. He wrote the program, so naturally, he should be able to answer any of your questions. [/Diu] Page 6 Mathematical functions calculators are built with a basic purpose: mathematics. Programming, playing games and everything other is secondary. Thus, you will find several powerful mathematical commands. Although they seem to be no job of a programmer, programs sometimes need mathematical functions, and many mathematical functions can be used in clever ways. In this guide we will group the commands in the following five categories: Algebra Symbolic Break 68k (TI-89, TI-92, TI-92+, and V200) is the primary coolelement of the calculator. With the solution () command, the calculator can correctly solve the appropriate number of equations (of course, it is almost even easier to get solutions). There are a dozen variables on the solution as well as a few commands to extract different parts of an expression, which should be useful for writing your own algebraic tools. As on earlier calculator models, there are also logs and complex number operations, which are also better with symbolic mathematics. The complete list of algebraic commands is: The advantage of 68k calculator for basic calculations is that it can calculate correctly up to 256255-1 with the correct numbers (and approximately floating point decimal calculations up to 101000-1). Here is the complete list of calculation orders: +, -, *, /, ^√(), !, , ▶, 0b, 0h abs() approx.() ▶Bbin roof() comDenom() ▶Dec E exact() floor() fpart() is very useful for calculations as compared to other models; they can calculate symbolic differentials and integrations as well as unlimited amounts and products. There are some digital functions as well as the first ones that are taken from calculator models. The complete list of calculation orders is: ∫(), ∏(), ∑() arcLens() d() deSolove() fMin() fMin() impDif() 3.10 Limit() nMin() nInt() Tailor () Data is a field in which 68k calculators are not prominent compared to other TI models. In fact, the data tools on the TI-83 series calculator are quite low (compare their page on the data). What remains is mostly a type of regression model: for all of them, use the Showstate command that shows the results in the dialog box, and look at the statistical system variables for more information. There are also commands for some general purposes for sample data: Finally, you can plot data with the NewPlot command (see Also Plotson and Plots Off). The basic thing to remember when doing trigomenotre triger is to be aware of what angle mode you are in. By default, you are using the readin, where a full circle measures 2π. The other two angle modes are degrees, where The full circle is 360, and the gradiures (ti-89 on the new OS version for the TItanium and The Seafaring 200) where a full circle measures 400. The commands that actually work with them include the normal treage function (half of which — mostly in the infruse half — was included in OS version 2.07) and their immotics as well as the recticual coordinates (x,y) of the polar coordinates (r, which are also commands to convert into polar coordinates). There are also high-bolic functions (one is the equivalent of a high-bolic for each of the normal treage functions). As far as symbolic mathematics is concerned, you can use tSwide() and tCos() to rearrange complex effects using sin() and cos(), and hope to make them somewhat easier by doing so. The entire list of the metric commands is: Page 7 metrics and their commands are a rectangular grid of matrix elements. On the TI-68k calculator, the matrix can be a mixture of any skilled (unlisted) variable type that is valid in expression: you can have data metrics, wire meters, truth values, or expressions of the matrix. You can even mix and match variable types — one element has a string and a number in the next is all right. There are three ways to insert metrics on the calculator: using the [] pencil[] brake: for example[a,b,c] [d,e,f] (it is a matrix with 2 rows - row a,b,c is above row d,e,f). [] Use of brackets and semicolons: for example {b,c;d, e,f] { } using brackets: for example {{a,b,c},{d,e,f}} (do this because metrics are actually saved as lists of lists) You are a particular element of matrix Access the element you want then enter in the [] bracket: Matrix[r,c] will access the element in the rth RO and the cth column of the matrix (the matrix is always indexed first by row,) from top to bottom, and by the other column, from left to right). Also, using an index — matrix[r] — the rth row of metrics is returned as 1 through the #matrix. The matrix on the first calculator models had a random access feature: the same time was taken to access any element of the matrix. This was possible because the metrics were restricted to the number. On the 68k calculator, since metrics can mix element types, they are no longer random access: The calculator has to go through the entire matrix to reach one element, so the larger an index is, the longer it is to access it. It is not important for small meters. But time continues to grow linearly, so access to the last elements of a larger matrix can be very slow. Except for the tightness of free memory, and the time of access to elements, there is no limit to the number of elements in a matrix. In matrix as directional mathematics, a vector n is a list of numbers that contain sinus representation in n-digital space (two representations, actually: as a point in n-space, and as a translation that takes the original to this point). 2- And 3 digital directional directions are used for 2-money space (airplane) and 3-money space, in general, in order to be used. On Calculator, metrics with only one row or one column are interpreted as directions for the purposes of commands dotP(), the pass p(), and unit V(), plus the formatting commands ▶Cylind, ▶Polar, ▶Rect and ▶Spyre). Linear algebra operations general mathematical orders and operators are spread across metrics in linear algebraic manner. +, -, and *, for two metrics, the related metrics are processes (especially the matrix multiplication is quite complex). ↑ Multiply a square matrix by itself and increase the correct number to power; if the correct number is negative, it requires text first. Matrix is a special operator for himander: T, called transpose operator. It flips the matrix about its main vein, so rows become columns and column rows. Operators +, -, *, and/can also be applied to a square matrix and a skeyler, multiplying the scaller with the iD matrix. For multiplication and distribution, this results in the process of each element, while addition and minus result in the addition or differential of the peatosin in each element on the central sector. Since sometimes you want to factor these operations factor by element, alternatives .+, .-, .*, ./, and .^ are provided, which do this for both two metrics and one matrix and expression. The exponential and tremmetric functions are a special explanation of the exponential and tremmetric functions applied on the calculator matrix, e^() is the most common. These commands need a square and sour hand imputation to the matrix, and a nearfloating point value needs to be returned. A is a better hand-crafted matrix A which can be displayed in shape A = PDP-1, where D and P are square metrics, and D is the turret — the main vetra plus is completely zero. If the matrix is a sire, the calculator can count clear values for D and P using Ag-VL() and egCV(): D = dif(egVL(A)) P = egCV(A). The calculator first applies functions like e^() to the matrix by writing metrics in the form of PDF-1 and then returning PF(D)P-1. Here, the function is applied to D by taking f() of each of the arterelements (elements away from the vetra sesame). This definition is used for the following commands: ^, ln(), log(), and root(), cos), sin(), sin 55(), tan(), and tan(). Other actions about the cosh (), cosh 55), Sanh (), sinh 55(), Tanoh(), and tanh), meters, most mathematical orders extend to meters, applying to each element; THE GCD () is a good example. Yet other orders are presented in unexpected ways. Sorting and sorting commands sortrows and column symmes as if they were lists. The following list of mathematics/statistics commands follow the meters as they do on lists of lists, resulting in a row direction containing the operation done on each column: Finally, there are commands that are specifically for meters. These metrics are found on the submenu Math Pop-up Menu. ID▶List▶List() mHou() mHouD() New MRouD() Routine () QR conditional statements, such as if, when(), and while, accept ingestion values as well as the only truth values. The check will be interpreted only if and only if every element of the matrix is true, by effectively blending each element of the matrix and. The most common way to create meters of truth values is with the mailmail operators (=, ≠, >, ≥, <, ≤) that apply to the metrics. They will return the only value 'false' unless both sides of the relationship have equal size metrics, in which case the matrix will be compared by the element, return a new matrix. Page 8 strings and their commands are a combination of letters with a string sequence (commonly called characters). They are most used to display text, although they can be adapted to any purpose. For the character table, see the Character Code page that can be part of the string. Use the text marks around the characters you want to contain to write the string. For example, hello is a string containing characters H, e, l, l, and o in this order. Using the String () command, you can also create a string from another variable type. On the TI-68k calculator, the string has an advantage over lists or meters - it's a random access structure, which means that access to the end of the string is as fast as accessing the end of the string (for the list, on the other hand, to finally go to an element, the calculator has to pass through every previous element of the list). Although strings are usually a cause of nomenclic for access, so lists are in short-slicloss compared to, the longer the string will beat lists and metrics, at least to access a specific role. Except for memory tightness, there is no limit on the amount of characters in the string. There are two special operators that can be used on the string - & (concatenum) and # (direction). The connection operator, &, includes two wires together - Hello& The World Hello world back. The nondirectional operator, #, automatically replaces the string containing the variable name - #f(x) is equal to f(x) (it can be very powerful, especially for images like non-algebraic variables). (Disturbingly, there is no easy way #ٹائپ ti-89. You can select it from the Character menu (second Cchar, 3, 3) or catalog) relationship operators (=, ≠, >, ≥, <, ≤) can also be applied to the string. It's easy to see how = and ≠ work: If two wires are made together with the same letters, they are equal. Alphabetical setting is used for other relational operators: if they come before it in the dictionary, one string is less than the other. But.. Not at all — there is a problem with the upper case and the small letters. This comparison is actually done by comparing the character code of each character in the wires, so all the smaller letters are considered to come to an end. کہ یہ وہ. خطوط cat <dog, as= you= would= expect= (because= cat= comes= before= dog= in= the= dictionary),= but=></dog,> <cat. the= tricky= aspect= of= strings= is= accessing= a= character.= to= do= this,= you= have= to= use= the= mid()= command:= mid(string,start,length)= will= return= length= characters= from= string,= starting= from= start.= as= a= special= case,= mid(string,n,1)= will= return= the= nth= character= of= string.= characters= are= numbered= starting= from= 1.= of= course,= there= are= several= commands= specifically= designed= to= be= used= with= strings= (though= not= as= many= as= we'd= like).= all= of= them= except= shift()= and= rotate()= are= found= in= the= string= menu= (press= 2nd= math= to= access= the= popup= math= menu,= then= select= d:string).= char()= dim()= expr()= format()= instring()= left()= mid()= ord()= right()= rotate()= shift()= string()= page= 9= code= timings= this= page= documents= the= speed= of= certain= commands= in= 68k= ti-basic.= now ,= obviously,= the= times= given= here= (in= seconds)= will= vary= from= model= to= model= and= even= from= calculator= to= calculator= (due= to= battery= levels,= free= memory,= and= other= factors).= however,= one= thing= that= does= not= change= is= the= relative= speed= of= the= commands.= so= if= you= come= here= to= see= if= if= then.. else.. endif= is= faster= than= when(),= the= information= will= be= useful= on= any= calculator.= elsewhere= in= this= guide,= you= might= see= assertions= like= foo()= is= faster= than= bar()= without= any= reason= or= proof.= the= information= on= this= page= is= the= reason= and= proof= behind= them.= testing= format= for= each= test= done= on= this= page,= a= variation= on= the= following= function= was= used:= timetest(n)= :func= :local= t,i= :starttmr()→t= :for= i,1,n=></cat.> <command> :End1000)تقریبا: برائے{clocTmr(t),1}/n) :EndFunc اس فنکشن کی ان پٹ n، جو یہ ہے۔ اوقات کی تعداد کے لیے لوپ کرنے پر حکم اس میں بھی زیادہ وقت لگ جائے گا۔ اس فنکشن کا آؤٹ پٹ ایک فہرست ہے جیسے {185 5} ان میں سے پہلا وقت کی پیمائش ہے: پیمائش کی جانے والی چیز کو کتنے ملی سیکنڈ کام کرنے لگتے ہیں، ان میں سے دوسرا اس پیمائش کے لیے ایک نقص حاشیہ ہے۔ نقص اس لیے <command> بھی بڑا ہے، امتحان بھی اسی طرح درست ہے۔ تاہم اس میں بھی زیادہ 185±5 ملی سیکنڈ لکھا جائے گا۔ ان احکامات کے لیے جو بہت کم وقت لیتے ہیں، یہ متبادل شکل مفید ہو سکتی ہے۔ اس میں اس وقت کو Tmr() اور پڑتال stintmr() واقع ہوتا ہے کے سیکنڈ کے پورے نمبر استعمال کرتے ہیں، تو اصل وقت دونوں سمت میں صرف ایک سیکنڈ کے تحت بند ہو سکتا ہے۔ نیچے دی گئی جدولوں میں اس طرح کے نتیجے کو 185±5 ملی سیکنڈ تقریبا(End1000: برائے timetstret(n) :مقامی: فنک s,t,i :sattmr()→s :i,1,n :Endبرائے :cattmr(s)→s :sattmr()→t :i,1,n :End1000)تقریبا: برائے{cetmr(t)s,2}/n) :EndGunc اپنے ٹیسٹ وں میں حصے ڈالنے میں خود کو کوڈ ٹائمنگ کے ساتھ تجربہ کرنے کے لئے مفت محسوس کریں <command>اور اپنے، پیش کیا جانا ہے جو کے لئے.. اینڈ برائے لوپ خود چلانے کے لیے لیتا ہے۔ نتائج کو اس صفحے پر ڈالنے کے لئے تاہم، ایک ہی وضع اور طریقہ استعمال کرنا یقینی بنائیں! اس Also, list calculator models and OS versions (found in F1, About A: ... Menu). Unless otherwise stated, all the test ships on this page were conducted with 200 calculators and OS versions 3.10. It's for details and explanations. </command></command> </command> now the originals come The lube and conditional when () command is often an easy alternative to a j... and. Endif blocks when only one value needs to be stored. It's small. The test attempts to find out if it is too fast. When the code test was done (if the code test was done :if the condition is :1 :0 :Endif both tests were done with a correct as well as incorrect condition. When codetime (ms) (), true bet 8.2±0.05(when), false bet 8.15±0.05 if.. Other.. Endif, true bet 8.5±0.05 if.. Other.. Endif, false bet 8.65±0.05 Result: When () the order wins, but not from many people (although there is a difference that cannot be explained by mistake only). Also the results of the right and wrong situations are very close, they are probably equal. Although variables () and # (nondirection) work differently, they overlap substantially. If you have a string with a variable name inside, you can be used to get value for this variable, both #and more (). (If you are a dissuadad, the difference is that #, unlike exchange(), can be used to point to variables, not just its value — but can be used for the value of a whole expression stored in a string). When tokenized, both commands take 2 bytes, so there is no difference in size. The question is, which is faster? Code time (miss) #14.35±0.05() 17.05±0.05 Result: When you can# is better than you can use. List vs. String vs. Matrix This test attempts to compare the time required to access elements of lists, strings and metrics. Before the test, the three variables were initialised for the competitive values: this list was set to 2048 elements, all 0 (as the correct number). The string was set to 2048 spaces. The matrix was set to 32 rows of 64 columns (total 2048 elements), all 0 (as the correct number). In each case, elements have one byte of data. However, only silk stores them all with a pad and only one header; as a result, the list and metrics variables are larger. It is known that when the string is stored internally as an array, lists and meters are stored as an expression in the taurus polish edify. As a result, it is expected that access to any element of the string will require equal time; access to the list or matrix element will take longer as the element will be further away from the start. Although it was not tested, the total size of the variable should not significantly change the access time. List string matrix codetime (MS) CodeTime (MS) CodeTime (MS) List[1] 9.1±0.05 mid(str,1,1) 40.5±0.1 m[1,1] 9 .8±0.05 List[500] 36.8±0.2 Mid(str,500,1) 41.8±0.1 m[1,64] 13.35 ±0.05 List[1000] 64.5±0.5 Mid(str,1000,1) 44.3±0.1 m[32,1] 115±1 List[2048] 122± 1 Mid±str,2048,1) 49.6±0.1 m[32,64] 118±1 Result: Initially, exactly, access to list or matrix elements is faster than the string character. However, the cost goes up substantially: for around 600 elements, silk will win. Of course, to store digital data, The list will have an additional advantage, but in the long run (say, for 800-1000 elements, by method), the string will also be better for it. If this is possible, you may also consider using several smaller lists instead of a large list. As for metrics, it is clear that the cost of time increases much faster in going down a column than across a row. This can be a factor in how you choose to turn your data around when it doesn't really matter. You probably will not improve the time tomorrow, but the proper trend will ensure that time is not dramatically different. << Improve Review Release Programs >> Page 10 RoutineS This page is a combination of short routines that can be used as subroutines in large programs. On the contrary, the Sample Program page links to all sports (which may use some of these routines). Among the examples of techniques is still a third type of program — short programs that may be of some value to themselves, but are primarily here to demonstrate the use of some programming techniques in an original program. Please follow these instructions when participating in the routine: A routine must be self-made and relatively low in length. A routine is well tested to ensure that it works right. Add inc:routine as the starting point for pages. When the normal page is over, link to it here. Page 11 is a common name for a small image displayed on the Sprite Spirite screen (usually, this image is also moved around, or repeated several times like a tile map). In TI-68k bisc, support for spareties is in construction: The picture file can be any size (up to the total size of the screen) and can be displayed at any location using commands such as RCLPI. Spritey is always rectangular; even if the picture itself is a circle, there will be empty pixels around the circle to square the spreit data. Spright logic There are four basic ways to show a black and white spart. The easiest thing is to write more of what was there at this point before. Alternatively, you can apply AND, OR, or XOR logic to each pixel in the background and the sparet to produce the result. RPLCPC uses the overwritten logic to display the sprit. This is often useful when you plan to cover the entire screen with spareties, for example. The flaw is that it can erase the bits of other things that are drawn on the screen. XOR XorPic uses xOR logic to display a spirite. This means that the pixel is flipped on the screen for each deep pixel of the sparet. XOR logic is the easiest to use for sparety, since THE XOring will erase it at the same spot a second time and restore any pixels under it. However, if the sparety is often going to overlap (against the occasional), then XOR spareties may not look this great. Or RclPic uses logic or to display a spirite. With OR logic, pixels can be set, but never cleaned. Many potentially overlap For those on screen, or pulled with best looking spareties. However, they are not like this To erase (you can store the background before spreat drawing, but this can have a very complicated code). And and the PIC uses logic to show a spirite. And with logic, pixels can be cleaned, but never set, so it's limited use by itself. It can be used when drawing in white-on-black or as an equivalent of logic instead of black on white. Also, and logic is useful in mask spareties (see below). Mask Mask is a modern technique that allows sparetes to be transparent in some places and non-transparent in others. To do this, you use two sparets: self-sparet, and a support sparet, called 'mask', wherein the result is transparent wherever you want, is dark pixel, and otherwise light pixels. To display the media sright, you must first and support the sprite, or xOR (either one works) important sprite. Like any spreit technique other than XOR, maqad sparets destroy the background below them, so if you ever want to erase the sprate, you'll face problems. One solution: If you only have a sprite screen revolve around while the background remains the same, you can save the background in a screen-sized picture. Then whenever there is a need to finish the sprit, you do rplcpc while eliminating the background. Another solution: For multiple sprate, you area for a picture under The Stapak Sprate, then that part of the RplcPic background when you want to erase the sprate. This can freely de-mail these spareties, but is complex to manage (the easiest way to use direction). Tile mememing tile maps are a widely used technique for drawing maps in sports such as R.P.G. The idea is to use small (e.g. 8x8) sparety in grid-printers to cover the entire screen. Each area has its own spart (e.g. grass, water, brick wall) associated with it. Similar idea can be converted into side-sclanging games such as Mario. The easiest way to store tile map is in the correct number matrix;

to easily change the sparetes from these exact numbers, the name of the related spareties is 'Tile1', 'Tile2', ... And so, or something similar (then you'll get nth sprite #(tile&slic(n)). Tile map can be shown in for two pencils. End for Lops: :for row, 1, height :for col,1, width :RplcPic #(Tile&slic(Map[row,col]), 8*Row-8,8*col-8 :Endfor :Endfor (This code assumes 8x8 sparets, tile maps stored in matrix 'maps', and protected sparets in images 'Tile1', 'Tile2', etc.) uses four () instead of a more efficient method string () — see the four () command page for more information. Scrolling will look better than showing each screen as it comes out if the map can be scrolled. It is possible to use a combination of stopak and RPLCPC. Use StoPic to save all but one row or column of the screen to a temporary picture. Then the RPLC Pic they have the same picture, but end a row or column. Finally, the rest row or column with tiles Use the help to write. Here's an example that left the scroll (rv0 and col0 col0 Variables used to track where you are in the map::StoPic tempo,0,0,8*height,8*(width-1) :RplcPic tempo,0,8 :col0-1→col0 :For row,0,height-1 :RplcPic #(Tile&sring(Map[row0+row,col0]), 8*row,0 :End because even more modern techniques are smooth scroll. Here, the same idea applies, except that you move the screen to one pixel at a time. Also, because you can't draw negative screen coordinates, you may have to do some slyly when scrolling left or above, or just not use the screen full. Improving XorPic This technique is useful when using XorPic to show a dynamic spirit, especially when it can move in just a few directions. The basic idea is that xOR operation is associated with the bits: (Screen XOR pic1) XOR pic2 = Screen XOR (pic1 XOR pic2). The common code form using XorPic to display dynamic spirite is below: XorPic pic1,row,col © Erase old location Row+1→row © Update Text lessons XorPic pic1,row,col © To improve this, create a new picture which is a combination of pic1 and pic1 is moved down to 1 row, using XorPic. You can think of this picture as the pixels that need to be changed so that pic1 can be moved down to 1 row. If this picture is saved in pic2, then the better code is as fornows: XorPic pic2,row,col © update spareate row+1→low © update coordinates This technique is that the advantage of showing a spareate is twice as fast as showing two. On the other hand, it needs two images displayed. In addition, the code can be complicated if more than one direction is possible. Stocking pictures can end up using a large, graphics-intense TI basic game many photo variables for sparets. Although you can release these photos with your program, it can be disturbing because the user has to transfer all these photos to the calculator. A better way to use the installation program that makes these pictures. There are three ways: a reference picture This method includes only one photo file that includes all the sparees next to each other. The installation program remembers this picture on the screen, then uses a stupak to store the spareties to separate the variables. The advantage of this method is that it is small and fast; but it is not as aesthetically happy as it may be because the user will have to see the reference picture during installation. This method uses a newpic to store each spireat individually. This method avoids drawing towards the screen, and is still fast. However, one flaw is that Nupak uses a matrix to store the spareties, which have two elements for each deep pixel of the sparet. Even smaller spareties take a lot of space in the installation program. Encoded Newpic To Fix It, You Can Try To Encode The Spratees Differently (An Example Using A Large Correct Number — A Single Correct Number Can Store 254 Bytes Data, Which Is Good For About 2032 Pixels Of Any Spirette More than enough). Then, the installation program creates metrics for The Newpic Bee. This fix the only loss of the newpic, the size. However, another disadvantage appears: Making metrics is a slow process. << Dialog Review Animation page 12 >> Lists and their commands is a set of elements in a list setting. On the TI-68k calculator, lists can contain any variable type of mixture that is valid in expression: you can have lists of numbers, string lists, truth values, or expressions. You can even mix and match variable types — one element has a string and a number in the next is all right. Only one particular case is list lists: they are allowed in a way, but they are called meters, and there are some more restrictions on them. Lists are written around the elements using the curdi brake { and } , separated by a kama. For example, {1,2,3,4,5} is a list of numbers from 1 to 5 in this order. You can access a particular element of the list by writing the number of elements you want in the [] bracket then: Listwar[5] will select the 5th element of the list. The number of elements starts at 1. On the first calculator models, the lists had random access features: the same time taken to access any element of the list. This was possible because the lists were limited to numbers only. On the 68k calculator, since lists can mix element types, they are no longer random lying in reach: The calculator has to go from the entire list to reach an element, so the larger the index, the longer it is to access it. This is not important for short lists. But taking the 100th element, for example, almost double the time it took to take the 1st element, and the time continues to increase linearly, so it may be too slow to access the last elements of a long list. Except for the tightness of free memory, and the time of access to elements, there is no limit to the number of elements in the list. Actions on lists Many of the orders, including math orders and others, can be extended to lists by applying them to each element of the list. There is an example that (): sin({1,2,3}) {sin(1) sin(2) sin(3)} If there are more than one argument in an order, there are two ways to extend it to apply to the lists. One is to use it with a list and a regular argument: then the order will be applied to every element of the list that is paired with regular argument. Thus, the modern ({10,20,30},7) {3 6 2} is as follows. The second way is to make lists of both arguments. In this case, the size of the lists must be the same and each element in the first list will be paired with the related element of the second. For example: mod({10,20,30},{7,8,9}) {3 4 3} Although modern () in these examples can be expanded in both ways, sometimes only one is possible. For example, PTON can be used with two numbers or two lists, but not with a list and a number. On the other hand, the round (), can be used with two numbers or list and a number, but will give a meaningless expression when In two lists. A remarkable special case is the basic operators of mathematics (+, -, *, /, ¯, and ^), which can be used (both the way) with all the lists. List-specific commands are, of course, specially prepared commands for use with lists. Many of these, such as fade (or rotate), can also be used with wires. Many of these commands are found on the List menu (press Second Math to access the Popup Math menu, then select 3:List). Add() The Press P() Low SMS() Dim() dotP() exp▶list() Fill Left() Δlist() List▶Mit() Donot▶List() Max() mid() Minutes () New list() polyEvil() products() rotate right() () seq() senda sendD sum() lists can also be applied to multiple statistical orders. Accept lists of statements that are subject to lists, such as if, when(), and while, truth values as well as single truth values. The cheque will be interpreted as true if and only if every element of the list is true, by effectively blending each element of the list and. The most common way to create lists of truth values is with recell operators (=, ≠, >, ≥, <, ≤). They will return the only value 'false' unless both sides of the relationship have lists of equal sizes, in which case the lists will be compared by the element, return a list. Page 13 Glossary Argument is one of the values that a command follows. A combination of digital variables arranged in the list or matrix to be processed by the array calculator. Each element in an array is referenced by its index - position in the array. ASCIAI Us Standard Code information exchange It uses 1 byte (8 bits) to keep a character, and thus it is limited to 256 different characters. The calculator uses an internally modified version of ASCII to store characters (see Character Code), but the TI-Basic spokesperson is related to the token instead. In addition to Assembly TI Bisc, ti graphicing calculators are to be built into other programming languages. It's very difficult to learn and program, but at the same time it's much more powerful, and it can be used to create very fun games and programs. You can also set up C in the Assembly program. Binary two digit (bit) number system is based on 0 and 1, just as the normal decimal system is based on 0-9 numbers. For example, the number in the binary 10101 denotes 1*24+0*23+1*22+0*2+1 or 21 in decimal. To enter the number in the binary on the TI 68k calculator, type 0b and then type the number. Bit a binary number (0 or 1). Deviation from the orderly performance of branch program statements. An unconditional branch calculator causes a single branching statement to be run over a specific location in the program every time it faces a branching statement. A conditional branch transfer program control based on the result of a logical state. Breakpoint is a point in a program where program implementation can be suspended, used when you fix the program. In TI-Bisc, such break points can be created with text commands. Bug error in a program (see De Buging). Byte a string of eight-bits. The calculator uses a byte of information to uncode letter A, as well as most tokens. Bytes (or kilobytes - one kilo byte is 1024 bytes) are used to measure the amount of a program or variable on the calculator. (A popular programming language on many platforms. C programmes can be set up in the Assembly for ti 68k calculator using TIGCC. Character a character, number, punctuation symbol or a sign of special graphics. They are encoded internally in asciai format, but the TI-Basic spokesperson uses tokens instead. The clock cycle unit of time in cpu. It is so small that even the most initial assembly instructions take several clock cycles to run; the amount of clock cycles for the same TI-bisc command can be in millions. Order a word or token that will ask the calculator to do something. Examples: For RclPic, at most(), etc. they are sometimes subdivided into functions and instructions. Ampersund — and — is the connection operator. For example, TI-&primary=TI-basic. Condition A logical expression that the calculator can assess to be true or incorrect. A constant specific number, list, matrix, or string that is given as a fixed value in the program (such as 9.024 or hello), rather than in a variable. The brain of the CPU (Central Processing Unit) calculator where the calculator controls the processing and implementation of everything. Cursor displays a small, glowing square where the typed character will appear. The cursor (in the form of a cursor) is also used on the graph screen to put a point from the user. Data information, often digitally in shape. When data is used through a calculator program, it can be hard coded or soft coded. Fix programs to remove bugs, or errors. There are many technical for de-buging, some universal, some are specific to calculator programming. A 10 digit system using decimal 0-9 symbols. This is the default number system used by the calculator. Default is a standard attribute or value that the calculator assumes without customising. For example, RclPic displays a picture on text (0,0) by default (top left side). It can be written more with more arguments to specify text uadstosk to display. The message box displayed on the screen for dialog input and output. Enforce another name to run the program or command. An exponential number that shows the power to which a number or expression is to be raised is usually written right and above the number. For example, 26 = 2x2x2x2x2x2x2x2. Scientific explanation: The power of ten which is multiplied by mantisa to calculate the original number. Flash ROM/Archive memory on the TI-68k calculator where applications are safe. Format used in TI Bisc to store floating point decimals. It uses scientific information to show all the numbers. Flowchart a sketch Arrow-connected shapes that show the progress of a program. Flowchart works to create complex programs and give an example of how programs work. The verb word has many meanings in the context of TI-Bisc programming. This can mean a type of command that returnvalues - such as the gcd() or sin(). Or it can mean a type of user defined program that returned the value. There are any kind of visual displays on the graphics screen, such as graphs, footnote drawings, both stationed and dynamic. Hard code is the information used by the hardcoded data program that is from the logic of its orders. The program is very easy to read, but it may be impossible to change. Hardware circuit boards and other electronic parts that make up the calculator. Hexagon 16 symbols, 0-9 and a-F using a 16-digit system. It is used as an easy shorthand way to express the binary code, because every four bits of the binary have a related hexagon symbol. To enter the number in the hexagon on the TI 68k calculator, type 0h and then type the number. Index position in an array. TI-Basic sinus sits an index of 1 starting from 1 (so that the first element in a list is 1, second 2, and so on). In some other languages, their index is started at 0. Input sources through which data is inserted into the calculator by the user, mainly through the calculator keys (although link port is also a form of input). In the case of directive TI-basic programming, it is often used to indicate a type of command that does not return value, but stands only: for example, line, or input. It is sometimes commonly used to describe commands. The correct number is a whole number, positive or negative (or 0). Typically, ti-basic seisc has a number that is definitely the correct number, which means that all its digits are saved perfectly. You can also use floating point numbers for almost calculations with decimals. Algorithms stored within the interpreter calculator that run the TIBSk program by implementing proper system routines. A recurrence of a lup. For some commands, the calculator has an internal loup for which the total number of recurrences is relevant, and affects accuracy. A panel of keys used to enter programs and data in the keypad calculator. A list of variables that can be accessed and modified from their location. Lists in TIBscan have a mixture of variable types. Type a group of one or more continuous program lines that are performed repeatedly, either in a certain number, or until a condition is fulfilled. The basic digital part of the number appeared in the Mantisa Scientific Tote. Mantisa in 3.264E + 4 is 3.264. Two-money grid of matrix variables that can be accessed and modified from their position. Lists in TIBscan have a mixture of variable types. Memory two different locations (RAM and ROM) where calculator programs and data are saved. All variables and programs take part To store, that's what's important in the size of the program or variable. The MHz calculator run at cpu speed. It is from 10 MHz to 16MHz on the TI68k calculator. It is measured in millions of clockcycles per second. A way to write a number system. There are two common systems of decimal (Arabic) systems and Roman numerator systems in the use of people. The calculator uses the binary system internally, but the decimal system is what it normally displays. A symbol used in operator calculations (digital operators) or relationship comparisons (related operations). Math operators +, -, *, /, ^. Recellal operators >, <, =, ≥, ≤, ≠. Logic is the runer and, or, xor, no. Improve the process of improving the time, length, or memory usage of a program. Output information sent from the calculator, such as graphics on the screen. In addition, the sources through which the data calculator leaves, either through link cable, graph link, or USB cable. OS (operating system) the internal program that is running the calculator and includes all the functionality needed to use the calculator. List of program instructions that tell the calculator what to do to perform a task. Programmer A person who writes the program. Programming language can perform and digital or alphabetical commands that the calculator can understand. RAM (Random Access Memory) is a temporary memory, i.e. the one in which data is saved as long as electrical power is applied. The data in the RAM can be accessed or changed and lost if the batteries are removed from the calculator. Recurrent is a program that invokes itself as a subroutine (or a function that is defined in its own way). See The Rectore. Some instructions for rom (read memory only) calculators are permanently saved to ROM and can be accessed but cannot be changed. Data in rom is not lost when batteries are removed. Scientific explanation (scientific explanation) a method of expressing very large or very low numbers using ten times the multiplication of ten times (mantesa) raised to a force (exponent). Change the screen part to give you the fill of viewing only part of the larger picture from the screen. SoftCode is the saved information in an array at the beginning of the softcoded data program, and read from it later. Reading is generally slow compared to hard-coded data, but the advantage is that the same code can often be used for different data stored in the same way. Sprite Is A Small Picture That Is Moved Around The Screen Using Code And/or Repeated Multiple Times On Screen. Statement A line of a program that contains the same instructions such as text, RclPic, max(), etc. Usually the command is equivalent. String is a variable type that protects text as a character series. Subprogram seqt a program class that can be used more than once during the implementation of a program, such as a complex set of calculations or print routine. You can create subprograms using the Explanation command. TI (Texas Devices) TI Graphicing Calculator Maker. In They also make a large rating of other electrical devices to graph the calculator. The unofficial name of TI-68k TI callicloters with 68k processors: TI-89, TI-89, TI-92, TI-92 Plus and 200. This is the | BD. The programming found on the TI-BISC TI calculator is the calculator described in the unofficial name of the language. Token tokenised TI bisc has a same instruction or symbol that takes 1 to 3 bytes and is used to uncode TIBSk programs. When the user is talking about the impact of a program or command, it is the person who run the program or command on his calculator. Variable a value given name that may be different during program implementation. Variable is a memory location where values can be changed with new values during program implementation. Page 14 Dialog Although input on the I/O screen (with commands such as input and input ster) may be useful for some programs, the dialog boxes are generally more effective. The dialog box is a common message box that is displayed on top of the screen that is currently on, but without writing too much (therefore the background is restored when the dialog box exits). The basics can be created by inserting a dialog. Dlog block ending in the program. The dialog will be displayed when that part of the program is run; after the dialog is deleted, the program will resume after the end dialog. Dialog only: EndDlog will give an empty dialog box, and actually causes an error. Use the following commands to provide content for the dialog box: Title title to dialog. Maximum can be a title. If the title is not there, the top of the dialog box will be empty. Add a line of text to the text dialog. Text must be stringed, so use the String () command to display another data type. The request add the text entry box to the dialog. The application contains two arguments, a line of text and a variable to store. Add the dropdown menu to the dropdown dialog. Dropdown takes three arguments: a line of text, a list of wires for dropdowns, and a variable to store. You can use text and application suo-mail, give the dialog box that has no other elements. Top details There are two buttons at the bottom of the dialog box: EST=OK and ESC=Cansal. If the ESC key is used to exit the dialog, the dialogue is actually canceled: The variables that may have been affected by requests and dropdowns remain as they were (generally, undefined). Fortunately there is a way to check if it happened or not. System Variable seok is well- calculated the way the last dialog exited. It does this in somewhat unusual way: If INT was pressed, it is valued at floating point value 1.0, and if pressed ESC, it is valued at 0.0 (perhaps it is a holdover from the TI-83 series calculator, which thus stored all truth values). Variable ok is not affected by the text instruction outside the dialog. EndDlog (which can also be exited using ESC, It has no two buttons), nor through the Pop-Up menu. Through. Works when the request directive is used automatically outside the dialog. End dialog. The AlphaLock request always secures a string on its variable, but is intended for all types of input. As a result, it has the option to close alpha lock, if you are put a number. Add a third argument to request to do so that 0 review. This functionality was added only with OS version 2.07, which can be a problem. It also does nothing on the wide screen calculator (since they have a QWERTY keyboard and do not need alpha lock). However, it still exists on the wide screen calculator for compatibility, and does not affect just anything. Unfortunately, if there are several applications in one of the dialogs. EndDlog blocks, they cannot have different alpha lock settings. Either they all have alpha lock or none of them (setting is used to order the first request, and the rest is ignored). Compatibility If the text in one of the elements of the dialog is too long to fit into one line, the dialog diamation errors. However, the allowed length of text differs from ti-89 and TI-89 to the titinium, in the wide screen calculator. This way, a dialog working on one calculator may error another. You might think that more text fits on the wide screen calculator, but actually it's not, because they use a different font for everything except the titles. The original limits are given in the following table: Command Wide Screen (TI-92/+/V200) Standard (TI-89/Titinium) Title 50 Four Approx. 39 Chars Text (Dialog) 38 Char Approximately 39 Char Text (Plain) 34 Four Approximately 37 Four Application 20 Four 20 Four Dropdown 35 Four Approximately 35 Char Note: Approx. That is, the actual upper limit is different because of the variable width of the font used. For dropdowns, the upper limit is at the total length of detail and longest option. It makes sense to write with the dialog box text that fits both sets of needs at the same time. Then, there's no danger of the dialogue ever being a flash! Default values If the variables stored by the application or dropdown are not undefined, they are used as default values: For the application, if the value of the variable is silk, it is already inanded and prominent in the application box. For dropdowns, if the variable value is a number, the relevant option is the one that is initially selected. This menu may be useful for changing options. Alternatively, you can use it for instructions or specific default values, for example: Enter the answer here→Request response response: If the wrong type of variable is not displayed, as if it was undefined. Dialog pieces These are some common code pieces that include dialogues that can work in almost any program. Of course, text can be made to your needs. Confirm :D Yaalag: The title Are you sure? Text: Unsaved data will be lost! EndDlog : If OK=1 Then : © : © : Endif Load/Save File :D ialog : Title Load File : Request Folder, Fold : Request Variable Name, var :EndDlog :If ok=0 : © Set :#(Fold&\&var)→File © File to save file, instead :file→#(fand&\&var) It is a good idea to ask the folder and variable name separately, because it clears questions about the composition. In addition, if you move: main→ fold in the first part of the program, a folder that was loaded from once will have a default value in the dialog. In fact, for this purpose, you want to keep two variables to save and load files, to allow different default values. Error Dialog : Try : © Central Program Code :Other : © If an error occurred : © Handle expected errors, then : Dialog : Title Error : Text An unexpected error occurred. : Text Error Code: &SLAK (Error) : Continue text program? : EndDlog : If ok=1 then : Clrr : Goto street : © sleat up vars : Password : Endif :EndAtret Such a dialog box will ensure that your code does not have the same bad result as a bug otherwise - there is a professional procedure to handle errors. You also want to add something to the message such as <email address=> mail with the Bug Report. Replace the pass-router with something like The Clrer:Leave Goto if you don't want to see the error message to the user, in any case. < assembly= overview= sprestes=>> page 15 Click here to edit the contents of this page. Click here to edit individual sections of the page (if possible). See the edit link headlines when available. Unedit the entire page source. See how this page has developed in the past. If you want to discuss the contents of this page - this is the easiest way to do it. View and manage file attachments for this page. Some useful tools to manage this site. See the link to this page and the pages that you add. Change the page name (URL address, possibly category). See the wiki source for this page without editing. View/set the Parent page (which is used to create a karbhome and structured layout). Notify administrators if this page contains objectionable content. Does something not work as expected? Find out what you can do. General Wikidot.com and Help section. Wikidot.com terms of service - what you can do, which you should not do, etc. Wikidot.com privacy policy. Page 16 Errors This table lists all possible error messages that may occur. The error number on the left is not shown, but is stored in the error system variable, which can be used in an attempt. Endpoint block. Some error messages are very clear (e.g. #810), but some are low. Where the error message itself gave little information, this table explains the error more well. A large portion of the information here was taken from the TI-89 Manual's Apandex B. Error Number Error Description 10 A function did not return value 20 A test did not resolve TRUE or FALSE This error usually happens when comparing an undefined variable, such as if. 30 </email>Folder name cannot be 40 Argument Error 50 Argument Mismatch two arguments must be the same type. For example, PTOn can be used with two numbers or two lists. But a number and list cannot be used together. 60 Argument must be a bolin expression or correct number for use with logic commands such as or can be applied to two truth values, or two correct numbers. 70 Argument must be a decimal number 80 Argument must be a label name 90 Argument Must be a list 100 argument metrics 110 Argument A Pic 120 argument should be a Pic or with the title used in the String Toolbar. Pic part only apply to the wide screen calculator, 89 or 89 titanium 130 argument must be a string 140 argument must be a variable name It can also indicate an invalid variable name such as 1xy. 150 The argument must be a blank name folder if they are not empty, the folders cannot be deleted. 160 Argument must be expressed For example, zero (2x+3=0,x) is not valid because 2x+3=0 is an equation. Use zero (2x+3,x) instead — =0 is inuse with this command. 161 ASAP or viable string Too long this error has been caused by the limit of assembly program RAM. The limit does not exist on the HOW1 calculator; on others it is 8k with AMS 2.03 or less and 24k with AMS 2.04 or more. If you try to run an assembly program that exceeds the limit, you'll find this error. 163 Attribute (8 number number) Object (number 8) not found 165 batteries Must be less than 170 bound lower bound upper bound to send/receive product code. For example, this error may be occurred with zero finder on the graph. 180 Break-on Key was pressed during calculation or while running the program. This error cannot be caught by trying normally. Endblock, unless it is in text. 185 Check III Error 190 Circular Definition Scan Circular definitions of a variable are caught: for example, a+1→a (if any undefined). But the round definitions of the function are handled by a range of recurrent depth. 200 Barrier Expression page for invalid page | For more details (with) operator. 210 Data type is an argument type of incorrect data type. 220 depending limit The same variable cannot be used as both a merger variable and a bound. For example, ∫(sin(x), x,0,x) will not be allowed. 225-Df EQ Setup 230 dimensionan index went beyond the limits of a list or matrix. For example, the list[5] when the list is equal to {1,2,3,4}. Some 240 dimensions of unmatched commands need their own list or matrix arguments to match in size. For example, you cannot add lists {1,2} and {1,2,3}. Divide 250 to zero, James Anderson! 260 Domain Error Some commands accept numbers within a certain range only. For example, ans() only works with 1-99 numbers. 270. If the immutable variable name is invalid if 280 other and other if other. Then ok block, actually, try.. End attempt blocks are also used by others. But you didn't hear me say that. 290 Endtry Similar Other Statement 295 missing Recurrence occurs when an aittyo solver lasts a long time without finding a solution. This usually means there is not one but really horrible cases. 300 Expected 2 or 3 element list or matrix It is with commands that deal with 2 or 3-money directions. 307 Flash application extension (function or program) not found 308 Flash application not found 310 nSwell's first argument should be a univariate expression English translation: the only undefined variable that can be in the expression of nSwell (is what you are solving for. The first argument of 320 solutions or solutions should be equations or inequality. Of course, the problem of zero () is exactly the opposite. 330 Folder Sometimes the calculator likes to be laconak. This error is on the VAR-LINK menu if a variable is stored in the No Folder. 335 graph function y1(x). y99(x) Not available in The Df Equation Mode 345 Non-Permanent Units 350 Out of the number range 350 Index 360 Direction string is not valid variable name 380 invalid ans() that many responses have not yet been stored. 390 Invalid assignment 400 Invalid assignment value 405 invalid axes 410 Invalid command 420 Invalid folder name 430 Invalid 440 Invalid invalid indiuated multiplication for current mode settings A(b) is used for function calls only, not to multiply a and b. If there is no fixed function, the calculator assumes that you have tried to multiply, and it errors. 450 invalid user defined functions in function or current expression cannot change global variables, or use certain commands. It also happens in the mails, such as data/matrix editors. Invalid in 460 customs. The EndCustom block 470 in the dialog is invalid. The EndDlog block in the toolbar is invalid 480.. In the Attempt, EndTBar Block 490 is invalid. Endblock 500 invalid label label names have the same limitations as variable names. 510 invalid lists or metrics lists can be only 1 amount (lists) or 2 amounts (list of lists). Lists of lists are metrics, and no non-recissive goods (e.g. {1,{2,3}}) are allowed. 520 invalid outside custom. End-customize or toolbar.. EndTBar blocks 530 invalid outside the dialog. EndDlog, Custom.. End specific, or toolbar.. The EndTBar blocks 540 invalid outside the dialog. Invalid invalid outside endDlog block 550 outside function or program 560.. For end-of-the-day, End for, or while. End while block 570 invalid path name 580 invalid polar complex You may think it is a prohibition against building military base on the North Pole. In fact, it is used with command. For example, (i-2) is invalid. 590 Invalid syncax block dialog with miscellaneous errors.. EndDlog, Custom.. End-last, and toolbar.. End T Bar Blocks. Invalid 600 Table 605 Units Abuse 610 Invalid variables in local statement scan not be localised, for example, by invalid variables. 620 Invalid variables or function names are being used for variable name building in-function. 630 Invalid Variables 640 Invalid Directional Syncax 650 Link Transmission 665 Matrix No. If 740 disappears, I'd like to. Endif Block 750 is not a function or program named 765 No function selected 780 No solution found Most commands usually think of any tricky to return, instead. This error occurs only with the contius graph tools. The non-algebraic variable in 790 expression 800 is the only real result if the calculator is in real number mode. 810 Not enough memory to save the current variable. Please delete unnecessary variables on the Var-Link screen and reopen the current OR as a reopen editor and use F18 to clear the editer. The possible range of 830 Tahar Swim Floating Point Number is between -101000 and 101000 (not included). However, sometimes a talaoh is replaceed with limits. 840 Plot Setup 850 program not found 860 recurrent recurrent 255 calls is limited to 870 deep protected names or system variable systems cannot be deleted, for example. 875 ROM Resident Routine unavailable 880 Setting up 885 Signature Error 890 Single Matrix 895 Slope Fields need a selected function and is used for 1st-ord equation slot only 900 Santi statistics. The 910 Syncax is an expression or entry just does not understand. For example, 2+2+. 930 Very few arguments 940 too many arguments 950 too many subscripts are much more than two. 955 Very much undefined variable 960 undefined variable 965 Unlicensed product software or flash application 970 variables in use or flash application is locked 980 variables, saved, or archived 990 variable names 8 characters 1000 window variables 1010 Zoom 1020 Internal Error 1030 Secure Memory Violation page 17 68k TI-Basic for 83 TI Basic Programmer This tutorial means introduction of 68k TI-Basics for programmers who are already quite experienced with TI-83 series core programming. This tutorial highlights differences between the two languages rather than re-education many things. A large newness of the 68k calculator has the potential to make symbolic calculations. It has many applications: You can deal with expressions such as x2+2*x+1 but consider x unknown, or deal with the correct value of √π/3 without having to sit with floating point values. The calculator no longer has some statistical orders, but it has too many powerful calculus orders (it can, among other things, make symbolic derivatives, inseparable, and even unlimited amounts). It does not have programming applications immediately, although you can often find unexpected uses for these commands. On the other hand, a very programming-related difference is the arrival of fault-catching. Using try and end statements, your program can identify whether an error occurred, and possibly recover from it (or on) Display a specific error message). Also one of the highlights is how many images have been empowered. They can now be of any size and can be displayed with any logic. Real real-time multiplayer games are now possible with The SindCalc Command, something that was impossible on the 83. The more specific differences described below are considered to be running programs faster, and allow programming style closer to programming more serious language on the computer. Commands on TI-89 can be entered in letter and no need to be selected from the menu. In practice, programs and functions are tokenized, taking the command range in a program from 1 to 3 bytes. Many commands have been added or removed between the two languages (see the command index for the complete list of commands). Also, the following commands have changed in the property: there are two more changes in total. First, many command names have been cut off where they were longer than 8 characters: it is the maximum for the command name on TI-89. An example is RclPic which is 68k equivalent to the recall PIC. Second, the use of parentheses after the command is now followed by a strict convention. Directions — commands that do not return value — parentheses are not required (e.g. if, text, etc.) functions — commands that return value — parentheses are required (e.g. sen(), set mode (etc.) even working with arguments to use parentheses (e.g. key, starting Tmr() etc. Many orders have been added. However, as far as the data is in the range, 68k calculators are inferior, even less than the TI-83 series; most of the functionality is now reduced to regression, and the calculator internally does not know how to calculate most of the probability distributions. The way variables are stored has had major changes from the TI-83 series. Now all variables have a common name system in common: a variable can be named up to 8 characters long. Variables can also be placed in different folders, which cannot be made into a pencil, but otherwise the files on the computer are very similar to folders. By default, the variables are saved to the folder 'Main'. There are many variable types on the surface that were on the TI-83 calculator: you have digital variables, lists, metrics, strings, picture variables, equations, graph databases and a new one, called data. However, they are slightly different. Digital variables to start can now either change the floating point (same as the TI-83 series) or the correct number (which does not have decimal space, but has several hundred digits of accuracy). As a result of symbolic actions on the 68k calculator, you also have the following: formulas that are reviewed only as much as you can. The list is very different. They can now catch any combination of numbers, expressions, and character strings. This also significantly slows them down, but also slows them down significantly. Data are basically metrics, but with new capabilities (and speed) New lists. Although the metrics remained limited to the numbers, they kept their pace. You can effectively emulate the old-style list using only 1 row/column matrix, the bolin variables will also cause some confusion for Ti-83 programmers: The bolins (right/wrong) are returning one or zero sins instead of operations that can be used as regular correct numbers (1, 3, -5, 4.8 etc.), bolin variables are now a segment of all. This means that you cannot use a simple if (variable) statement to check whether a variable has been defined yet, and cannot use sharp options in expressions that include parentheses' bolin statements to see if a particular term is used in expression (e.g. x. (k>2)(x+1)+3x). Instead, you must use the ifVar() command to check whether a variable has been defined and return to regular if-again blocks for conditional computing. Programprograms are also considered variable, at any other level: You can also define the program within another program. They follow these commands, and they can be given parameters. Using local commands, you can announce local variables that are reset to their old values after the program is finished. You can also define functions that are similar to programs but return value. Functions have other limitations, though: they can only use local variables, and cannot modify any global aspects of the calculator (so, for example, the scripting commands are limited to programs). With local variables, and the ability to define functions and programs, you can program in a processor language style. Instead of keeping the entire program code in one block, you can divide it into functions and subprograms that are set up at the beginning of the main program. The whole issue of Memory Leaks (born on the TI83 series by being left out of the code blocks with Goto) is no longer available in 68k TI-Bisc. The loups offset the end from the beginning, so the program does not have to have a stack to be aware of what to do with the instructions. There is no longer any memory cost to enter a laptop (or any other type of code block), so it is impossible to leak the memory like this. Improve Most types of minor automation from the TI-83 series are invalid on 68k calculators. For example, closing parentheses, excerpts, and brackets is now mandatory — but don't add any sizeto the program, as it has been turned into tokenized and post-fax edited. Ans Variable no longer plays an important role: Although it has ans() command in place, it is not modified by storage of variables within a program, so it is mostly useful on the home screen. A large part of 68k automation revolves around careful use of lists. List variables are no longer random access: Access to the last element of the list is much sofer than access to the first element. For this reason, going from a list to a for-the-top is about the worst thing you can do. 68k gives the programmer a lot of options with graphics displayed. The sparetes, which had to be shown using various tricks or libraries on the TI83 series calculator, are built in 68k as image variables. These picture variables have little similarity to the screenshot-like functionality that they have on TI-83+. These can be any size from 1x1 to the entire screen, and can be saved and remembered in any part of the graph screen. There are even several commands to show a sprite using different logic. In addition to these most powerful orders, more general orders have also been set up. In practice, all graphics commands have one point and pixel equivalent, so you are free to choose one or the other to use (generally, you want a pixel). The circle command now quickly creates circles against taking several seconds. Instead of being forced to choose between the home screen and the graph screen, the selection is between the graph screen and the Program I/O screen on 68k callicloters. Program I/O is a separate home screen for screen programs, which is limited to text (but text must not be aligned). In addition, both screens can be spiced up using dialogue, which copies the appearance of the pop-up window on the computer, and is great for putting data without deleting anything from the screen. Another major addition to the graphics command set is the new created dialog feature. New commands add additional I/O capacity that does not interfere with program I/O or graph screen, which allows added data entry in the program. There is a limit of graphics on the 68k calculator - they cannot pull on the top menu bar and the status bar below, so are effectively limited to only 2/3 of the screen. Assembly libraries can be used to access the entire screen, but it is impossible only in TIBSk. End words This page provides a review of some of the features of 68k, but it is not complete, and cannot. You can also see more pages to get a better picture of 68k programming: Command Index FAQ Sample Program however the best way to try learning language is to experience first hand with it. Page 18 Key Codes This table contains values returned by each (key press, modifier) key () that is available for the couple. The TI-83 series returnkey press with 68k calculators' key() modifiers, unlike the key found on the calculator. On TI-92, TI-92+, and 200 seajourneys, the key layout is different. The same effect (e.g. F8) still is consistent with the same value (e.g. 275), but is achieved differently (by pressing F8, in this example, but also the other+F3). Alpha key is taken from the grebe button for arrow keys. Consider that the key codes for typing characters are consistent with this character character code. For example, the key code for typing A is 65 and Ord (A) will also return 65. It even applies to key codes for international characters that are not displayed in this table. Key Modifier No ⇧ (Shift) Second ♦ (Hiera) Result Value F1 F1 268 F1 268 F6 273 Y= 8460 F1 268 F2 F2 269 F2 269 F7 274 Window 8461 F2 269 F3 F3 270 F3 270 F3 275 Graph 8462 F3 270 F4 F4 271 F 4 271 F4 271 TblSit 8463 F4 271 F5 F5 272 F5 272 Table 8464 F5 272 ♦ (Hera) Copy 24576 Kit 12288 Alpha A Lock Escesc 264 ESC 2 4360 Left Paste 8456 ESC 264 apps 265 apps 265 Switch 4361 8457 apps 265 on 267 Home Away From Home 277 Home 277 CUST 4373 Home 277 Home 277 Mode 266 Mode 266 mode 266 ▶ 18 _ 95 Mode 266 Catalog CATLG 278 CATLG 278 i 151 ∞ 190 CATLG 278 ← (Backspace) ← 257 ← 257 INS 4353 DEL 8449 ← 257 Clear 263 Clear 263 Clear 263 8455 Clear 263 X 120 X 88 ln (4184 e^) 8280 x 120 Y y 121 Y 89 sin(4185 sin¯55 (8281 y 121 Z 122 Z 90 cos(4 186 cos¯55 (8282 z 122 T t 116 T 84 T 116 T 84 Tan(4180 tan¯55 (8276 T 116 ↑ 94 ↑ 94 π 140 It is 136 ^ 94 | | (with) 124 F 70° 176 Format 8316 f 102 ((40 B 66 { 123 b 98)) 41 C 67 } 125 © 169 c 99, 44 D 68 [91 8236 d 100 ÷ / 47 E 69] 93 ! 33 e 101 × * 42 J 74 √ (4138 & 38 j 106 - - 45 O 79 VAR-LINK 4141 Kontrst + o 111 + + 43 U 85 CAR 4139 Kontrast - u 117 INLET 13 13 Entry 4109 Enter approximately 8205 13 ST O▶ → 258 P 80 RCL 4354 @ 64 p 112 = = 61 A 65 ' 39 ≠ 157 a 97 EE E 149 K 75 - 159 S YMB 8341 k 107 (-) - 173 Space 32 ans(1) 4372 8365 Space 32 . 46 W 87 > 62 ≥ 158 w 119 0 0 48 V 86 < 60 ≤ 156 v 118 1 1 49 Q 81 34 8241 q 113 2 2 50 R 82 \ 92 8242 r 114 3 3 51 S 83 Unit 4147 8243 s 115 4 4 52 L 76 : 58 8244 l 108 5 53 M 77 METH 4149 8245 m 109 6 6 54 N 78 MEM 4150 8246 n 110 7 7 55 G 71 ∫ (4151 8247 gm 103 8 8 56 H 72 d(4152 8248 h 1 9 9 57 I 73 ; 59 8249 i 105 Arrow Key Modifier None ⇧ (Shift) Second ♦ (Heera) alpha/@ ⇐ (Left) 337 16721 4433 8529 33105 ⇑ (up) 338 16722 4434 8530 33106 ⇒ (right) 340 16724 4436 8532 33108 ⇓ (down) 344 16728 4440 8536 33112 ⇖ (left+up) 339 16723 4435 8531 33107 ⇗ (right+up) 342 16726 4438 853 33110 ⇙ (left+bottom) 345 16729 4441 8537 33113 ⇘ (right+down) 348 16732 4444 8540 33116 An easy way to find the press price without consulting this page is to write a short program to the output key code. : :P rgram :local k :0→k :while k=0 :glatKy()→k :End: End: Text string (k) :EndPrgm page 19 character code This page includes the character codes used internally by the calculator. This table can be useful, especially with four () and ord () commands. Characters can be used for variable names in the following ranges: 48... 57 (number, until they start variable names) 65... 90, 97... 122 (Normal alphabet, case inanimate) 128... 139, 141... 148, 181 (Greek characters, case sensitive: σ) is not like Ω) 192... 214, 216... 246, 248... 255 (international characters, case indifferent) be sure that value 2 (STX) cannot be used on the home screen, because for some reason it does not exist in the larger font. Value Four value four values Four values four values four values four value four value four 0. N.U.L. 32. Place 64. @ 96. ' 128. α 160. ... 192. À 224. Aka 1. SOHe 33. 65. A 97. A 129. β 161. ¡ 193. Á 225. á 2. STX 34. 66.B 98. b 130. Γ 162. ¢ 194. Â 226. â 3. ETQ 35. # 67. c 99. c 131. γ 163. £195. 227. a 4. EOT 36. $68. D 100. d 132. Δ 164. ¤ 196. Ä 228. ä 5. ENQ 37. % 69. E 101. e 133. δ 165. ¥ 197. Å 229. å 6. ACK 38. & 70. F- 102. f 134. ε 166. - 198. Æ 230. 5. BEL 39. 71. G 103. c 135. ζ 167. § 199. Ç 231. ç 8. BS 40. (72. H 104. h 136. It is also 168. √ 200. È 232. è 9. TAB 41.) 73. I 105. i 137. λ 169. © 201. É 233. é 10. LF 42. * 74. J 106. c 138. ξ 170. A 202. Ê 234. ê 11. VT 43. + 75. Of 107. k 139. ∏ 171. «203. Ë 235. ë 12. THE MINISTER OF STATE IN THE WORLD , 76. L 108. l 140. π 172. -- 204. E 236. 13. CR 45. - 77. M 109. m 141. ρ 173.¯205. 237. 14. (Lock) 46. . 78. N 110. n 142. ∑ 174. ® 206. Î 238. î 15. - 47. / 79. O 111. o 143. σ 175. - 207. Ï 239. ï 16. 48. 0 80. P. 112. P. 144. τ 176. ° 208. 240. 17. ◀ 49. 1 81. Question 113. q 145. φ 177. ± 209. Ñ 241. ñ 18. ▶ 50. 2 82. R 114. r 146. ψ 178. ² 210. Wu 242. And 19. ▲ 51. 3 83. S 115. 147. Ω 179. ³ 211. Ó 243. ó 20. ▼ 52. 4 84. T 116. t 148. ω 180.¯212. Ô 244. ô 21. ← 53. 5 85. U 117. u 149. E 181. μ 213. 245. 22. → 54. 6 86. V 118. v 150. E 182. ¶ 214. Ö 246. ö 23. ↑ 55. 7 87. W 119. w 151. i 183. · 215. × 247. ÷ 24. ↓ 56 8 88. X 120. x 152. r 184. + 216. Ø 248. ø 25. ◀ 57. 9 89. Y 121. y 153. T 185. 55. I have 249. x 26. ▶ 58. : 90. Z 122. Z 154. $\bar{x}$ 186. o 218. Ú 250. ú 27. ↑ 59. ; 91. [123. { 155. $\bar{y}$ 187. » 219. Û 251. û 28. ∪ 60. < 92. \ 124. | 156188 ≥ □. d 220. Ü 252 ü 29. ∩ 61. = 93.] 125□157 { □ 221 ∫ .189 ≠. Ż 253. ż 30. ⊂ 62. > 94. ↑ 126. ~ 158. ≥ 190. ∞ 222. L'Kvid 254. 31. ∈ 63. ? 95. _ 127. ♦ 159. - 191. ¿ 223. First 255. i Page 20 68k Welcome to Basic Start Kit TIBasic Developer (TI|BD) 68k Basic Starting Kit! The tutorial is developed to help the new 68's programmer get feet from the ground. Each one that is divided into these chapters has its own attention, and has to read in order. If you have questions or are stuck, leave a post on forums and someone will help you. Happy coding! Page 21 Click here to edit the contents of this page. Click here to edit individual sections of the page (if possible). See the edit link headlines when available. Unedit the entire page source. See how this page has developed in the past. If you want to discuss the contents of this page - this is the easiest way to do it. View and manage file attachments for this page. Some useful tools to manage this site. See the link to this page and the pages that you add. Change the page name (URL address, possibly category). See the wiki source for this page without editing. View/set the Parent page (which is used to create a karbhome and structured layout). Notify administrators if this page contains objectionable content. Does something not work as expected? Find out what you can do. General And the Help section. Wikidot.com Terms of Service - What you can do, what you should do Etc. Wikidot.com privacy policy. Page 22 Planning Program Before you write any code for a program, you must carefully plan the program. It seems like an unnecessary step, time that can be spent better, but in the end it will yield a big profit. Planning not only results in better quality programs, but will also reduce coding time (since you don't have to waste time rewriting the program) — win-win situation! The first thing you want to do when planning the program is what the program will do. Early programmers often say they want to make a cool game, but they're not far away from it. They must have a real opportunity to create their own program, they need to determine what the purpose of the program will be and then build. For program ideas, see Project Ideas page. When an idea for the program comes up, you should try to be realistic about the limits of TIBSk, and what a program can and can't do. For example, a game that takes a lot of speed for the user to play, like The Fox or Mario, is really not very practical beyond just moving a few things at any one time in TIBasic. Once you can determine what the program will do, you need to decide what features the program will have. This can include possible program options, interfaces (on the Program I/O screen, or on the graph screen), menu, one about screen, user support, and anything you want. Coding will also be easier as you get more complete with your program planning; it's for you to do a good job. If you cannot offer any ideas for your program or you are not sure that if the ideas you have put out understand, you should get input from the TI community. The three most friendly, active user forums are: Maxcoders Revsoft Calcgames Cemetech Omnimaga CodeWlurs Since these are the kind of people who are going to use your program when they end, you want to ask them to review your program ideas and present some constructive criticism. They can also give you some new ideas you never thought of. Even if you plan a program well and get community input, it's not just possible to think about everything in front. When a program is in heavy development, there may be more to do than make those changes initially when making changes later, but if you believe the program will be better with change, there is nothing wrong with changing or modifying your plans. The next step in this process is changing program plans to a sudo code. The Peudocode includes the use of English (or whatever language) inplace of the TI-Basic code to describe what the program will do to perform the desired functions and tasks. This prevents you from getting stuck in a TI-basic recipe, which can give you a more clear focus on the program. Starting by seeing you see the big picture of the first program And then again It is down in small and small details. Using a sketch as a base, this means you will put the most important things first and then gradually add everything. This can mentally picture how the program is going to look and make sure you don't forget anything. A key part of creating useful sodo

codes is including comments across. It is very easy to get lost in your logic or create problems that you have no idea about solving. In addition to telling you what to code (i.e. simplifying coding), it will also make you slow and think through the logic of your program. Still, the comments are just as good as you. The same large program is quickly unused and difficult to manage. When you're still editing the program, it's better to put it in many smaller pieces (subprograms). You can re-assemble them in a program when you've worked. One of the advantages of this approach is that you can almost change sodocode in an important program right now. For example, imagine this sodocode program: Setup menu - User enter difficulty, etc. Start the Variables Main-In-The-Love: Player Movement Draw Player Enemy Enemy End Man-Lup If the player wins the game then show the game win message otherwise display the logmessage cleanup you can translate it almost directly into a basic program. How do we do this (note that we don't write any code yet): :p rogm () :P rgp :supp)© settings, variables etc.: Main menu() © user enter difficulty etc.: initvars() © the variables: While state=0 © 0=game is still going : Moveplayer() © Motion Players : Draw Dropper() © Players : Movement() © Move enemy Drweney() © enemy :Endwhile :If state=1 then © 1=won, 2=lost : winam() © Display Win message :Others : Hargame() © Display Find message :Endif :c Clean up © settings and variables :EndPrgm Is Significantly Reduced In Tokenization Process As Another Benefit - Only Your Changed Programs Need To Be Retokenized. In addition, you can test each subprogram separately. When you develop in writing the original code, you create and edit each individual program (for example, you create and edit the menu) and write menus in this program. Of course, if these subprograms are large enough, you can distribute them to your sub-programs. When all subprograms are finished, the program will work in 50 or so pieces as it is (so you can test for worms and tick individual programs). However, if you want to release your program, you might not want 50 smaller programs to be sent. There are two options to add programs to a larger program: go through the main program, and replace subroutines with their code - press the other RCL to remember the subroutines in the program, and remove their pergram and end-program tokens. It is perfect for subroutines that are used only once. Add line explanation at the beginning of the program And use another RCL to remember the subprogram after this line. Then, the subprogram will be created when the main program is run, and you can delete it later (or call it local). Talk about local variables: This is exactly the right technique for ready programs. However, it may be easy to use when debuging, so it's best to avoid it when you're writing a program now. Review of << Issing Programs Review Comment Code >> Page 23 Programming Commands is useful for programming each command on the calculator, but some have been shown for this purpose. These are divided into about three categories: commands to control the flow of a program, orders to manage variables, and orders that provide input and output. Control Flow Any program is just a sequence of commands; but generally, it's not enough to run and do all of them in order, just one by one. Changes must be made to this command: Some orders should be followed only in certain circumstances, others can be repeated several times, yet others may be in the case of a mistake. Control flow commands are those that determine this setting. The easiest control flow commands are Goto and Lbl, which include just the program. This is not always the best way to work, though, and there are many alternatives: conditional a normal situation occurs when some orders are dependent on fulfilling a condition. The following orders resolve this situation: If the statement merely states a condition on the order or block of orders. The alice used with if the statement condition is not fulfilled provides an alternative branch. The Elsif statement used with If allows for several mutually exclusive situations. Terms are generally created by blending the equation relationship with logical operators (and, or, xor, no) (=, ≠, >, ≥, <, ≤). The lupus lupes allow commands to be repeated frequently. The following types of loups are: The lup. End-of-the-lobe blocks only repeat forever. While.. End repeat the block until a condition is satisfied. For. Repeat the blocks in the fixed number of times for the end. To deal with these lops, we have a couple of support orders: the cycle goes back to the beginning of a laptop before time. The exit leaves a drop before time. Subroutine If there is a number of tasks within a program, the code can be removed to subroutine: this routine can be called again whenever needed. The subroutine seibasics in TIBasic are their own programs. These can be defined within a program with the defined command, and there are two types: Functions (definition with Func.. and Funk) returns a value, and cannot affect the overall state of the calculator. The program (definition with pergram.. and pergram) can do anything, but does not directly return a value. There are two commands specifically for use with subroutines. The return goes out of a subroutine, back to the program that told him. Stop exiting every program that is currently running, subroutine or not. The last resort is found by catching an error if an error occurs. A part of the program or the entire program itself can also be put into effort. Other.. End block. The post-Alice code will run only in error, and The Option of ClrRr is — let's go like nothing happened — or pass-routers — handle the error as usual, with the OS error message. Another part of variable management programming is managing variables. This means: to praise them — → to praise them. Deleting them — with Delvar, Dell Type, Dell Fold or New Tube. Their safety — with lock, lock, lock(), archive, Unarchive, or isArc(). Address name and folder — copy, move, new name, or new fold, set fold (). There is also a local command, which calls some variables local to the program they use. Input and output Finally, a program must be able to get input from the user and give some output in response. For sports, this usually means using graphics commands, and reading keys with the getkey () command. However, there are other alternatives. Program I/O on screen: A more symbiosis is to use the dialog, the dialog is in the End dialog box. Input here is fulfilled with application and dropdown. The text command displays text appropriately. A title also comes in use. There are some miscellaneous commands left. Custom.. End-lastand-time and toolbar.. Create the Toolbar menu with the help of both the EndTBar title and the item command. And Popup shows a pop-up menu. Input and output also refer to other callicutors to be contacted. There are five commands for this purpose: Page 24 Subprogram a long program can be complicated to manage when it's all in one piece. This is why it is often easier to write a code with an important program that does its job by running several other programs called subprograms. In fact, it is quite easy to start using this method from the planning stage. Of course, there are other benefits to the use of sub-programs. It is good to define a subprogram for a task that needs to be done more than once. Also, a subprogram can run a recurrent program if needed. Of course, you don't want to release an end program in many pieces, so at the end of the day, you want to include subprograms in the main program. If the subprograms are not reused and used only once, it is easy: return subprograms using only other RCL to the place where they are used, and remove their pergram. parts of end pergram . See the next section for what to do in more difficult cases. You can also explain a subprogram in your central program. This is done using a defined statement: :D effin subprgm()=Prgm : © Subprogram Code :EndPrgm It's easiest if all subprograms are defined at the same time near the start of the program, but it doesn't really matter where you do it as long as you define it before using the subprogram. It is important to realize that the subprogram so defined is exactly like a regular variable. what if Do nothing so later The program is running, the subprogram will keep on going, which is not usually a good thing. There are several ways to clean it up, as in other variables: use local to call subprograms local variables (e.g. local sub-program1, sub-program2). Use The Dealwar at the end of the program to delete subprograms (e.g. Delwar Sub-Program1, Sub-Program2). Give programs a letter name, and use NewTube at the end of the program. See Setup and Cleaning for more information about how to do this. << System Variables Review Protected Data >> Page 25 Sample Programs Programs linked to this page are completed (if simple) programs, source code, and description. They are as a way to put everything on this guide together into an example that tibasic programmers like you can do. For example, for readers learning better than clarity, these games are also a great way to learn the techniques described in this guide. These programs have been fully improved, except to interfere with the reading ability. For example, variable names can be easily converted into single letters — it was deliberately left invalid. Individual pages explain what will be done to end the work of the automation. The tinnall reser — a game of reaction in which you run through a twisty surf, trying to avoid being overrun. RSA Encryption — Implementation of popular public key encryption algorithms. Symbolic distribution — a function that distributes symbolic. Disable on the button [68k OnBLook] — Block using the button while implementing the program Page 26 Isyour program Iscurrent Many programming guide give you the best advice on programming but stop at this point when the program is finished, tested, and improved. After all, most people can manage to release some program in some way or the other. But in fact, an inexperienced programmer can continue his work well in silence and simple form, which people will only look at without being able to stop. This tutorial will tell you how to avoid it, and get your program all the attention it deserves. Where to release first, it's important to know where to go to upload your program to the Internet. Although you want to create your own website and release all your games there, only your program will not be realized. Sure, having your own site may give you some publicity, but the best way to feel your game is to release it on one of the larger program archives (or all!) ticalc.org of ticalc.org CalcGames.org is by far the largest (and most popular), but you are also likely to be able to take more time waiting to present your program there; the other two will not keep it in more than a day or two. What to release is more than just the program you want to submit. These are the elements you want to put together — some of them are called file archive websites optional, but they are mandatory if you want to To succeed. The program is (obviously) if you were programming on the calculator, you'll need to transfer the program to your computer to submit it. You will need software like computer cable, and TI connect from the calculator. If you don't know where to get it or face problems using them, see Linking. Now, your computer has one or more files from your calculator. If there is only one, then you have to go good. If there are several files included, you should mention who each file is for, in the study. A significant step in offering readme programs. If you've never done this before (and possibly even if you have one), read our tutorial on writing the study. Generally, the longer the shorter the better (if no one understands how your program works, it's worse if he has what he already knows is re-explained) — unless it's a five-act game, in which you can consider re-addressing unnecessary things. Generally, your program may be longer and better; you don't need much for the least, say, a quadrupting solver. Also, please do not read in Microsoft Word 7 file format! A .txt file is sufficient, and is actually recommended. The three websites listed above screenshot sit down to let you add a bit or dynamic screenshot of your program. It's very easy to do — watch the screenshot page make it — and goes a long way towards making your program good (if it's really good). An attractive screenshot will inspire visitors to download your program more than the most flowery ass. Show your program here at the most impressive. The title title will tell visitors what your program is. A common error is making the title like the 8 characters name of the program. Don't do that — the title is first seen by people and you want to clarify it. Of course, if the program is called Tetris () it's tetris (although grey scale tatres, if so, could be even better). But if the program is called quadhale (), please create the title Quadrtic Swrewer instead! Don't forget this detail! There must be three parts: what the program is about. Solves all quadruptequations on complex numbers. The program features the best. Grey-scall interface at a low size of 13 bytes! No one needs to. Flyb needs to do the right job. Also, put each attached file in its own folder. The first two parts are positive; the third is negative, but it's important (imagine if your program doesn't create a specific file before, it's done without warning. 99% of your users will be lost, even if it is explained in the study, and write negative reviews). You want to make this section as short as possible, and the best way to do this is to avoid the needs in the first place. Also, you don't have to mention the obvious thing in the first place: If your program is in the core sports section for Ghalib, you don't have to say anything about Ghalib. The program and study to put it together are .zip archive, a community wide standard. The file upload form (it's different for all websites, but contains the same basic information that you're entered) should be fields where you can collect everything. You may also consider including a screenshot in .zip its normal location. Here are links to the file upload form of all the websites that are on this page. The form of Tcclick.org Form CalcGmes.org Form Cemetech.net Form Marketing your program may start when you first think of your program, although many people will not take you seriously unless you have at least one basic engine to show for your efforts. Other good points on which to advertise the program include the beta testing period before it is released to the public, and of course when it is finally released. For more marketing tips, see our marketing tutorial. << Code Timing Review Planning Program >> >>