Gerald's Column
by Gerald Fitton

This is the fourth tutorial article of a short series which I hope will help get you started with PipeDream 4's custom functions. These custom functions can be run unchanged in Fireworkz for RISC OS and in Fireworkz for Windows.

In this article, in addition to expanding on the topic of variables, I introduce the topic of Repetition using For - Next loops using as my main example a fixed term, fixed interest rate loan. There is a second example in which the interest rate and the repayments can be changed from year to year.

The Fixed Term Fixed Interest Rate Loan

Most Hire Purchase loans are of this type. Double click on the file [Repayment] and both the file [Repayment] and the custom function file which it uses, [c_Balance], is also loaded into memory and displayed on the screen.

Look at row 6 of [Repayment]. You can change the 'Initial Balance' (the amount borrowed), the number of years, the annual interest rate and the size of the annual repayment.

The 'Final Balance' is calculated by the custom function "final_balance".

It is the amount still owing at the end of the last year. The assumption made is the usual one that the interest is charged at the beginning of each year and that the first repayment is made at the end of the first year.

The custom function calculates the interest to be added at the start of the year and adds it to the outstanding amount. The amount of the repayment is subtracted and the result is the amount owing at the start of the following year.

Variables

Let me continue my discussion of variables.

I know that there are many different ways of classifying variables. The one I shall use is the classification into 'global', 'parameters' and 'local' variables. They are so named because of the way in which they are used.

Local Variables

An example of this category of variable is the interest charged during any one particular year. It is the variable "interest_charged" declared in slot [c_Balance]A11. Outside the custom function there is no point in knowing its value. Such a variable is called 'local' because it is useful only within the context (the locality) of the custom function. It is considered bad practice (and in some programming languages it is impossible) to use a value taken by a local variable outside the custom function which uses it.

If, within a custom function, “function01”, there is a cell which calls the same custom function, “function01”, then there will be two versions of the same custom function in existence at the same time.

It is essential that the two different values of the local variable (held within the two separate versions) are not confused. The PipeDream programming language supports such multiple existences of local variables.

The technique of ‘function01’ calling itself is called ‘recursion’. PipeDream custom functions support recursion.

If a variable is useful only within a custom function then it should be declared as a local variable and given a name. You will see that I have declared three such local variables in slots [c_Balance]A10, [c_Balance]A11 and [c_Balance]A12. They are the balance owing at the beginning of a year (“starting_balance”), the interest charged during the year (“interest_charged”), and the balance owing at the end of the year (“end_balance”).

Some of you who are into programming, looking a little further down the custom function to [c_Balance]A17, will see a variable “loop_counter” and wonder why I haven’t declared that as a local variable! In my view it has to be a local variable so I ought to declare it. Well, I don’t know if you’d consider it a ‘bug’ or not but, if you do declare loop counters as local variables then, in certain circumstances, you get an error message. Try to run [Loop]. It calls the custom function “[c_Loop]test_loop_counter”. I haven’t worked out exactly what’s going on but I don’t like it. Anybody got any ideas?

Parameters

Another category of variable is a ‘parameter’. The parameter called “no_of_years” which appears in slot [c_Balance]A7 is ‘passed’ to the custom function from [Repayment]D6 as the third argument of the custom function. There is little temptation to change the value of this particular parameter from within the custom function. However, some programmers encountering the ‘repayment’ problem for the first time might be tempted to change the “initial_balance” (another parameter passed to [c_Balance]) at the start of each of the years.

A convention of good programming is never to change the value of a parameter from within the custom function to which that parameter is passed.

An even worse practice I have seen on occasions is to change a parameter from within a ‘sub routine’ called by the custom function.

The conventional way to handle the requirement to ‘increment the year’ is to declare some local variables and ‘pass’ the initial values of those local variables to the custom function as parameters. That is what I have done in this example.

Global Variables

A ‘global’ variable is one which exists outside the custom function.

Let me be as honest as I can be here. Many programmers do change the values of global variables from within ‘sub routines’ (in PipeDream the equivalent of ‘sub routines’ are custom functions). I don’t recommend it because I think it unnecessary; furthermore I believe that it can cause difficulties for other people who come along later wanting to use the custom function.

I believe that global values should not be used within custom functions.

Let me give you a good reason why.

I have a phrase for custom functions which contain only parameters and local variables. I call them ‘stand alone’ custom functions because you can call them from ‘anywhere’ (ie from any spreadsheet) - you can build up a library of such custom functions and they will always ‘work’ because PipeDream will not get confused about the value any variable has at any stage.

It is a temptation to use set_value([Calling_Doc]slot,[Custom_Fn]slot]) within a custom function to ‘export’ data from a custom function back into the calling document. The slot [Calling_Doc]slot is a ‘global’ variable so far as the custom function is concerned.

Including the name of the calling document inside the custom function ‘spoils’ the generality of the custom function because it can be used only with a calling document of the name used. Some innocent person might try to use such a custom function from one of their own calling documents and, because the custom function is not ‘self contained’, then an error will be returned.

Instead of using set_value(,) within the custom function I recommend that you use set_value(slot,“custom_function”) within the calling document. That way global variables are not required!

Repetition

A repeated sequence of commands is called a ‘loop’. In PipeDream you can repeat a sequence of commands in three ways. The first method, the one used in this tutorial, repeats the sequence a preset number of times. The other two methods involve conditional execution of the ‘loop’ an indeterminate number of times.

In this example the commands which are executed a fixed number of times can be found in [c_Balance]A18, [c_Balance]A19 and [c_Balance]A20. The named local variable “starting_balance” is set to the “end_balance” from the previous period.

The “interest_charged” is calculated, added to the amount owing and the repayment (a parameter, hence the use of the @ sign in @repayment) is added to calculate the “end_balance” at the end of the period.

For - Next

The loop is bracketed by the commands for(,,) in [c_Balance]A17 at the beginning of the loop and terminated by next in [c_Balance]A21.

The arguments of for(,,) are the “loop_counter”, a named variable which is initialised by the for(“loop_counter”,,) command. The second argument (in this case the number 1) is the starting value of the “loop_counter” and the third argument (in this case the parameter @no_of_years) is the final value of the “loop_counter”. You can use a fourth parameter, the amount by which the “loop_counter” is incremented by the next command at the end of each loop. The default (used in this example) is a step (increment) of 1.

Indentation

In order to improve readability it is conventional to indent the commands which are within the loop. You can do this by typing a space (with the space bar) in the function line at the start of the command.

I have indented the three commands which are within the loop.

Nested Loops

This example does not include loops within loops but, when such a construction is used you must use different names for each of the separate loop counters.

It is conventional in nested loops to further indent the ‘inner’ loop (by two spaces for the second, inner, loop and three spaces for a third loop inside the second one) in order that the reader (who didn’t write the custom function) can recognise quickly where each loop begins and ends.

Using Arrays

I shall deal with this concept in two stages.

The first stage is ‘passing’ arrays to custom functions. The second is ‘returning’ an array from a custom function to a calling document. Only the first part (passing arrays to custom functions) is contained within this tutorial.

Essentially an array is a set of many values which can be contained within one slot or known by one PipeDream Name.

Have a look at the cells [Repayment]A12E18 and [c_Balance]A29A49.

You can change any of the values of the interest rates, [Repayment]C14C18, and the repayments, [Repayment]D14D18. The final balance will be recalculated by the custom function [c_Balance]final_balance_02 which you will find in [Repayment]E18.

The set of interest rates is passed to the custom function as the range [Balance]C14C18 and received by the custom function as the parameter “interest:array”. The appropriate interest rate for the year is extracted by the use of the function index(,,) in slot [c_Balance]A43. Similarly, the repayment schedule is passed as an array and the individual repayment extracted by the index(,,) function in slot [c_Balance]A45 of the custom function.

Summary

There are three categories of variable: local, parameters and global. Values of parameters should not be changed within a custom function. Local variables should be irrelevant outside the custom function in which they are used. I don't like using global variables within custom functions (even though I know some programmers will disagree with me).

What I have called a 'self contained' custom function is one in which the only variables are parameters and local variables. Such custom functions can be use by anybody from any calling document (just like anybody can use the square root function) without knowing why it works. Furthermore, PipeDream will not get confused if the same name is used for local variables within different custom functions which are all 'working simultaneously'.

An array can be passed to a custom function as a named parameter. If the custom function is expecting an array then it will use the array as an array.

A sequence of commands can be repeated a pre determined number of times using a for(,,) - next loop. The "loop_counter" is a local variable but should not be declared nor initialised separately from its first appearance in the for(,,) command.

It is conventional to indent the set of commands which are repeated so that the structure can be 'read' more easily.

Communications

If you wish to write to me then please do so at archive@abacusline.co.uk.