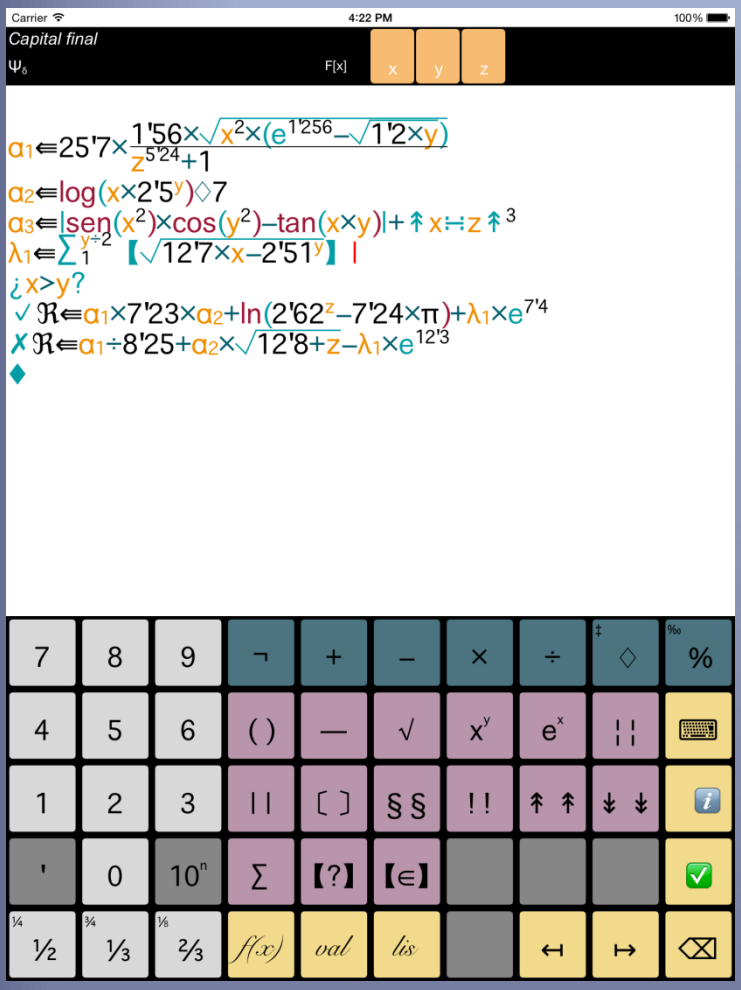


EUREKA CALCULATOR

MAHEMATICAL PROGRAMMING ENVIRONMENT

Powerful calculus engine naturally notated, for scientist and technological use

TCB CONSULTORES

MAHEMATICAL PROGRAMMING ENVIRONMENT

Eureka is a mathematical programming environment that is able to transfer the characteristics of the most sophisticated programming languages to your personal mobile device (SmartPhone, Tablet, etc.)

Eureka will prove to be invaluable to your work because of its:

• **Computing power**.

It comes with basic and advanced mathematical functions from the most sophisticated libraries.

• **Flexibility**.

It can be customised and fully adapted to meet your technical and scientific needs as it can be configured to show the functions that you need for your job and hides those that you do not.

• **Naturalness**.

Eureka uses a symbolic language designed and developed in-house called Magic that enables formulae to be written naturally, that is, it uses generally accepted mathematical notation and symbols, which makes formulations extraordinarily natural, easy to use and read.

• **Ease of use**.

Its natural language, symbolic representations, meticulously designed keyboard – divided into readily accessible and intuitive panels, its customisable settings and flexibility mean that the system is easy to use and quick to program, all of which makes it a simple and efficient tool.

OPERATING ENVIRONMENT

Eureka has two display screens: the screen for handling the repository, and the screen for programming and making calculations.

The first screen displays programming elements, which are stored in the database that makes up the active library: formulae, values, lists and templates.

This screen is divided into categories and elements. The categories are the classification entities defined by the user that enable programming elements to be grouped by topic, so that they are easy to access and manage.

The buttons that point to the formulation options are on the toolbar.

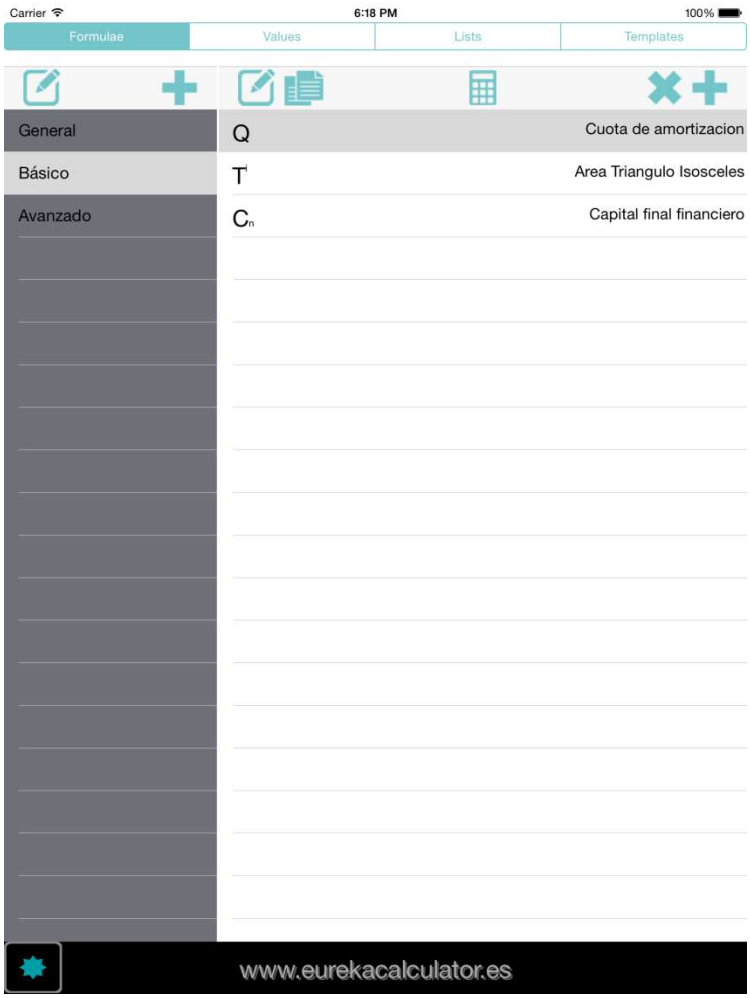
ELEMENT

The startup screen shows the repository of programming elements available, which can be managed from the buttons on the toolbar.

There are two types of entries.

• **Categories.** A category is a group of programming elements that serve to classify them by topic, usefulness, purpose or any other criterion that the user finds memorable. All elements must belong to a category. The category selected becomes an active category, to which any new elements created will be added. Once the elements have been created and assigned to a category, they can be moved to another category if the classification has to be adapted as the repository grows.

• **Program.** Eureka's startup screen displays a repository of the program's elements (formulae, values, lists and templates) so that they can be handled from there. The elements are classified by category and type. The following panels appear: (i) button for selecting the type of element; (ii) a categories panel for classifying the elements by topic and for handling them; and (iii) an elements programming panel for handling the various elements used in a calculation.



CATEGORIES

A category is a group of programming elements that serve to classify them by topic, usefulness, purpose or any other criterion that the user finds memorable. All elements must belong to a category. The category selected becomes an active category, to which any new elements created will be added. Once the elements have been created and assigned to a category, they can be moved to another category if the classification has to be adapted as the repository grows.

The following buttons can be found on the categories list toolbar:

Create a new category. By pressing this button a text box will appear that will ask for the name of the new category to be created. Enter a name and press 'Accept'.

Modify the text in the category label without changing the structure at all.

PROGRAMMING

Eureka's startup screen displays a repository of the program's elements (formulae, values, lists and templates) so that they can be handled from there. The elements are classified by category and type. The following panels appear: (i) button for selecting the type of element; (ii) a categories panel for classifying the elements by topic and for handling them; and (iii) an elements programming panel for handling the various elements used in a calculation.

• **Types.** The top bar shows the types of programming elements arranged as a radio dashboard with the following four buttons: **Formulae**, **Values**, **Lists** and **Templates**. One of the buttons will always be switched on to indicate the element selected and to which a new entry will be assigned. The elements belonging to the type selected are displayed in the list of elements (within the category selected). When a type is selected, the elements created in this type are displayed. When a category is selected, the current elements that belong to this category are listed.

• **Actions.** The lists of categories and elements each have a toolbar above them that enable the categories and elements of the type selected to be handled.

TYPES

The startup screen shows the repository of programming elements available, which can be managed from the buttons on the toolbar.

• **Formulae.** The formulae are mathematical expressions that make whatever calculation has been selected. They are either simple or multiline calculations. The simple formulae calculate a single mathematical expression. The multiline calculations break each expression up into various parts, so that the final result is obtained through the help of partial support calculations. Depending on the model, this type of formula also accepts conditional statements and loops (as commonly handled by advanced programming languages).

• **Value.** A value is a symbol that stores a numerical value. This value can be assigned at the time it is created, or it can be assigned as the result of a calculation using an assignment statement whereby the resulting value is stored in a multiline formula. There are special values, system constants and user values that depend on the calculation package used (interest, age, etc.).

• **Lists.** Eureka has intrinsic operators able to directly process lists as soon as they are entered. A list is a series of numerical values that can be accessed using an index value. The type of list establishes how the index acts to obtain the associated value. The list types are: ‘sequential’, whereby the index directly establishes the position of an element from a starting point of zero; ‘until’, whereby a search is made for an element whose reference value is less than the index value; and ‘from’, whereby a search is made for an element whose reference value is greater than the index value. The latter two are used to compile tables by value intervals based on the key value. The key value is the access code, that is, it returns the value of the element whose index coincides with the key value searched for.

• **Template.** A template is an interactive form, which pops up on screen and enables users to enter data and display results however they wish to see them. It has a screen on which headings, labels and open fields can be designed for entering data and displaying results, and a calculation pad on which the sequence for processing the data is set.

ACTIONS

The list on the right-hand side displays the elements stored by type and category. From the toolbar, users can:

- Create new elements in the type selected and assigned to the active category (selected). The programming and calculation screen can be seen from here.

- Remove a selected element from the calculation repository, once it has been checked that it is not used in any of the expressions of calculation.

- Access an element that has been selected and update its content on the programming screen if properly modified. The type to which an element belongs cannot be changed but its category can.

- Write any mathematical expression to obtain a result. This is done by pressing the calculation key, which will redirect the user to the programming screen.

- Duplicate an already existing programming element (formula, value, table or template) to create a new one. The user will then be redirected to the programming screen in order to change the element's name and content.

KEYBOARD

There is a keyboard with the symbols required for editing and programming the elements at the bottom of the programming screen. It is a dynamic, flexible keyboard that can be adapted for writing any given element or section of an element selected. It is made up of two panels, each of which can be viewed whenever necessary.

The standard keyboard has 50 keys divided into five rows of 10 keys each in the normal position (vertical) or into four rows of 13 keys in landscape mode (horizontal). A standard symbol is assigned to each key according to the configuration of the active dynamic panel. In addition to the standard symbols, a supplementary symbol can be added to each key. These supplementary symbols are displayed on the top left-hand side of each key and are activated by pressing the key for more than one second. Some of these supplementary symbols are fixed and therefore form part of the key on which they appear, whilst others are floating symbols that appear and disappear as required, depending on where the cursor is and the suitability of the symbol.

Special keys. Some keys have generic browsing functions that are valid for all panels. Firstly, there are the arrow keys for moving the cursor left or right. This is the only way the cursor can be placed in position. The alternative keyboard↓ key  (shown with the icon of a keyboard) goes from one panel to another on the keyboard (if this is possible). The **accept** key validates any data entered, carries out calculations or displays the menu for saving expressions. The special keys are in a different colour to make it easy to identify them.

The panels that make up the keyboard are:

1. **Alphabetical.**
2. **Numerical.**
3. **Basic calculator.**
4. **Extended mathematics.**
5. **New line.**
6. **Simple assignment.**
7. **List assignment.**
8. **Configuration.**

ALPHABETICAL KEYBOARD

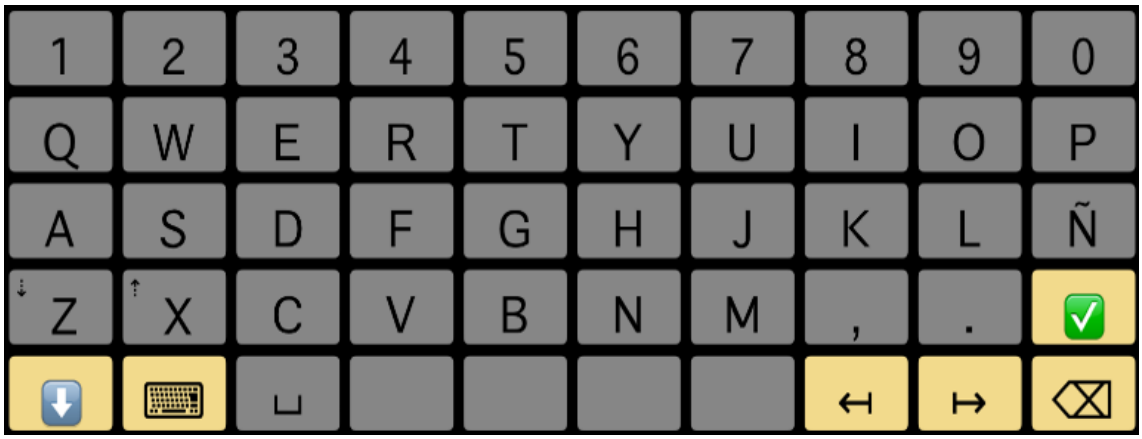
This enables users to write the symbolic names of the elements and their descriptive title. It is automatically displayed when the name of an element is assigned if it is being created or updated, that is, whenever the window assigned the name is enabled. It is made up of two different panels, the first with the Latin alphabet and the second with the Greek one. Thus, characters from both alphabets can be selected to create the names of elements and the calculation settings. There are four supplementary symbols on the keys at the bottom left-hand corner that can be used to enter subscripts (↓), superscripts (↑), join characters («).

 These keys change the keyboard from uppercase to lowercase and vice versa.

 This key changes the keyboard from the Latin to Greek alphabet.

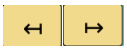
 These keys enable the cursor to be placed in the right place.

 This key deletes the character to the left of the cursor.

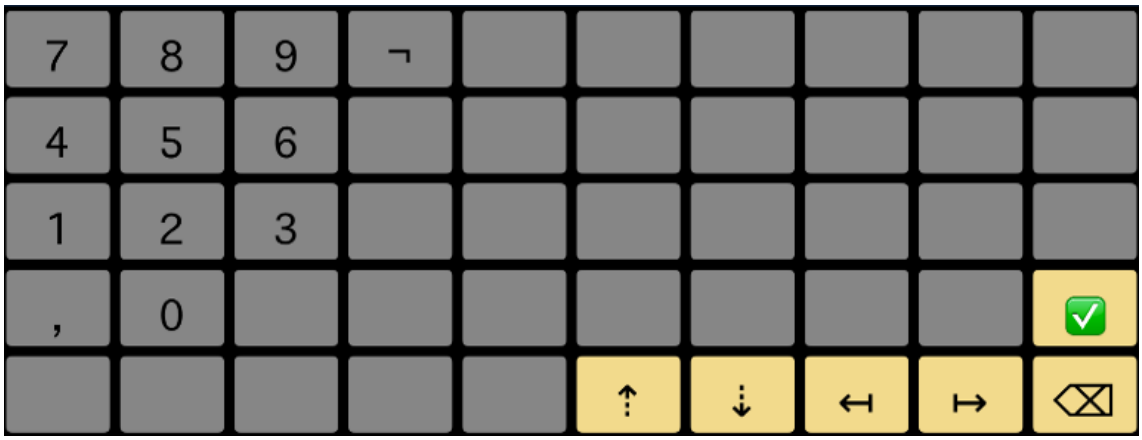


NUMERICAL KEYBOARD

The numerical keyboard enables users to enter the numerical values in the expressions. It is made up of the 10 digits (0–9) set out as a calculator, the decimal separator (,) and the scale, which makes it possible to express very small or very big numbers using the exponential notation (or scientific notation) by adding a power of 10 (positive or negative). Once the key has been pressed, the space for exponents (digits only) opens. If the key is pressed again, the exponent changes signs. In order to quit the space for exponents, move the arrow to the right as is the case for all other groups.

 These keys enable the cursor to be placed in the right place.

 This key deletes the character to the left of the cursor.

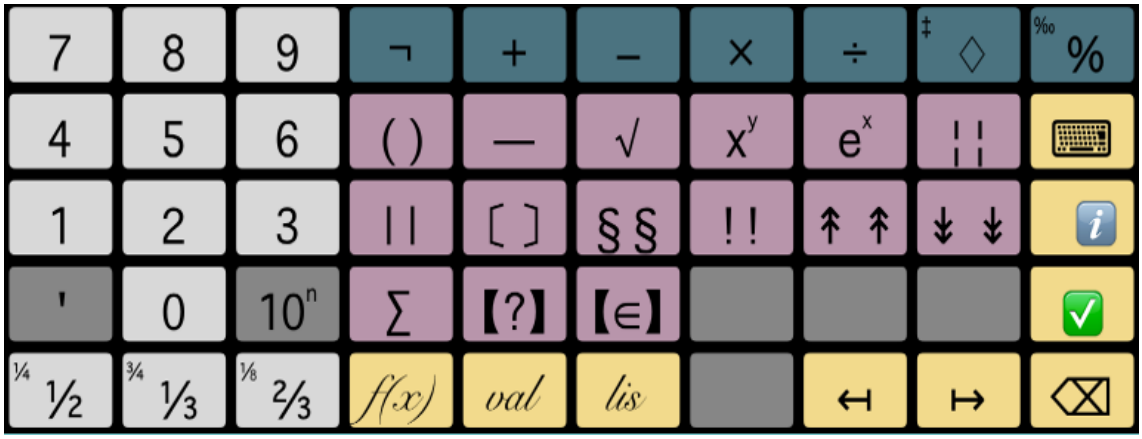


BASIC CALCULATOR KEYBOARD

This is the main keyboard used for programming formulae and simple mathematical expressions. It has the operators required for simple algebraic operations for groups and regular groups, and for the mathematical elements used in formulae, such as summation and decision tables. The first line of the mathematical keyboard is made up of basic operators that express simple arithmetical operations. (¬) Change of sign. (+) Addition. (–) Subtraction. (×) Multiplication. (÷) Short division. (◊) Module or remainder of an integer division. (%) Percent. Depending on which part of a formula is being programmed, the auxiliary operators or values are temporarily superimposed on the main keys in one corner of the screen. The operators (‡) Division quotient or integer division (result in whole numbers) and (‰) Per mille are displayed permanently in one corner of the screen. When a conditional statement is being programmed (conditions, conditional loops, conditional decision tables), the following corner screens are superimposed on the operator keys: (=) equal to, (≠) not equal to, (<) less than, (>) greater than, (≤) less than or equal to, (≥) greater than or equal to, (∧) 'and' true, (∨) 'or' true, (¬) 'not' true.

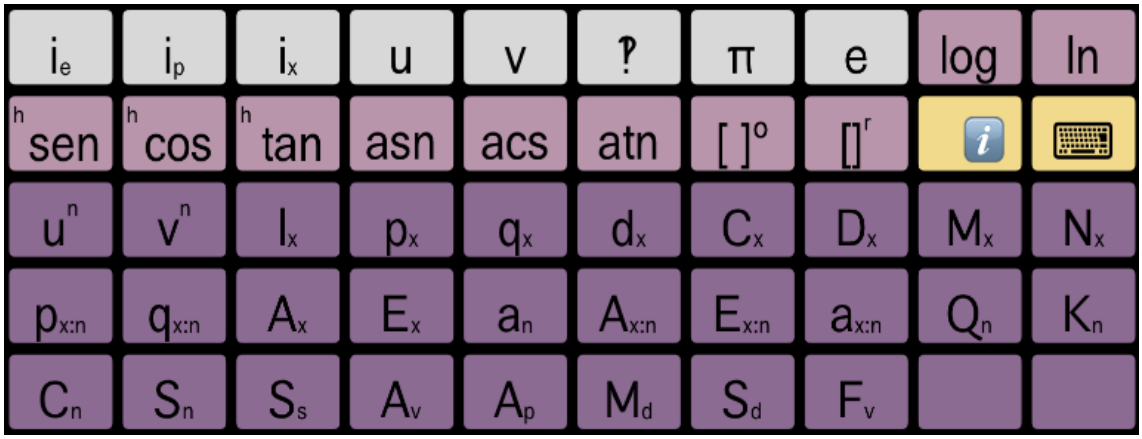
 This key switches from the basic to the extended keyboard.

 These keys show all the formulae as existing values and lists.



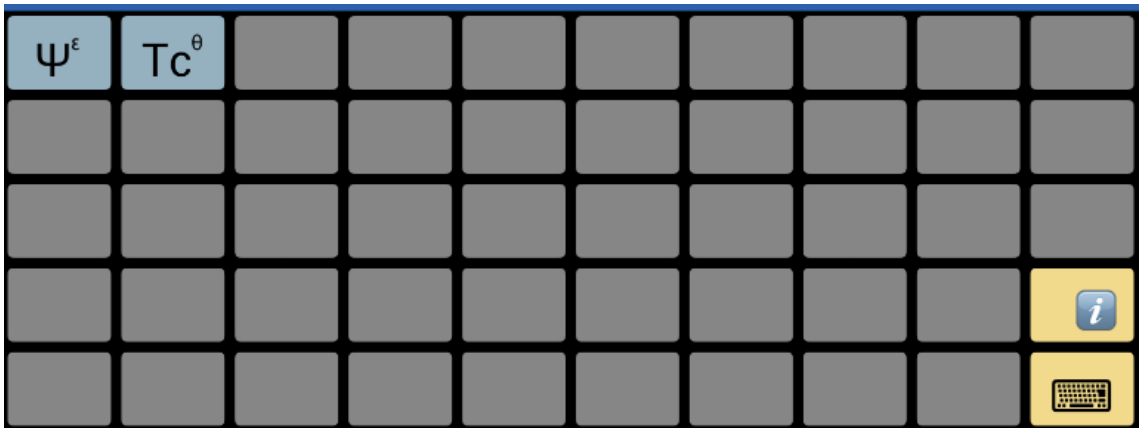
EXTENDED MATHEMATICAL KEYBOARD

This is made up of a second panel from the programming phase of formulae with specialised mathematical functions, according to user settings. The calculation constants, the logarithmic and trigonometric functions, the statistical, financial and actuarial functions, and, in general, the mathematical packages that are dynamically added to 'adjustments' can be found here.



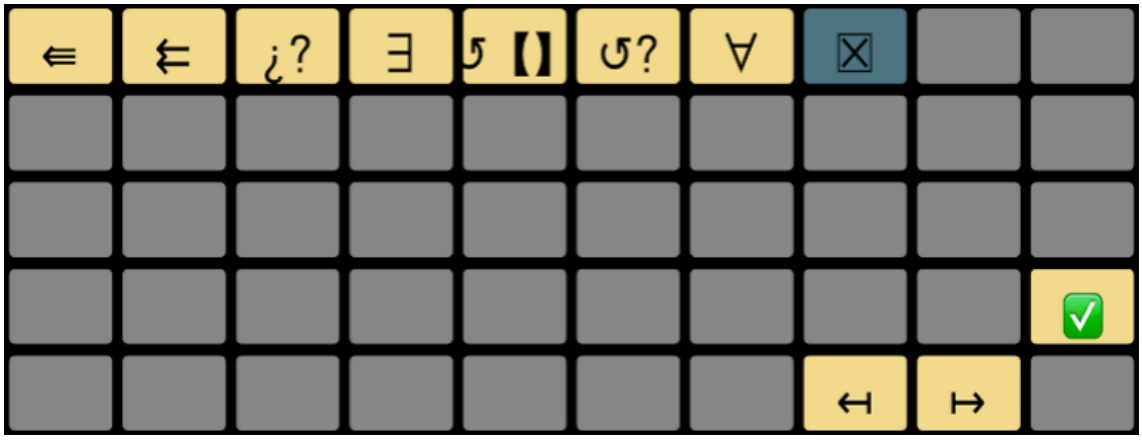
SYMBOLIC KEYBOARD

Agrupa los elementos de programación definidos por el usuario y que pueden incorporarse en las fórmulas. Se presenta un panel por tipo de símbolo (fórmula, valor, lista). Permite incluir de forma recursiva símbolos definidos en las fórmulas que los utilizan



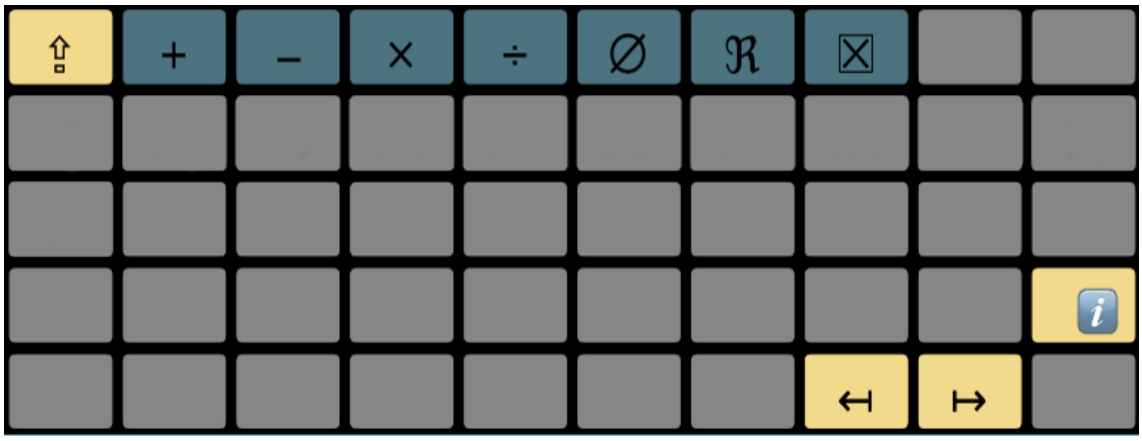
NEW LINE KEYBOARD

When the cursor is placed at the beginning of a line, this keyboard makes it possible to include program lines in front of the current position. The special keyboard for inserting a new line appears when the cursor is placed in a line insertion position, or when a request for a new line has been made on the assignment keyboards. The new line is inserted in front of the line on which the cursor is placed (unless the end of the program has been reached, in which case the line is added at the end). Each key corresponds to a type of line that can be inserted. ⇦ Assignment of a simple value, ⇇ Assignment of a list, ¿? Logical condition (yes or no), ∃ Option, ↺ [] Loop of values, ↺ ¿? Conditional loops, ∀ List loop. The symbol (☒) also appears, which is used to delete the whole block (according to the type of line).



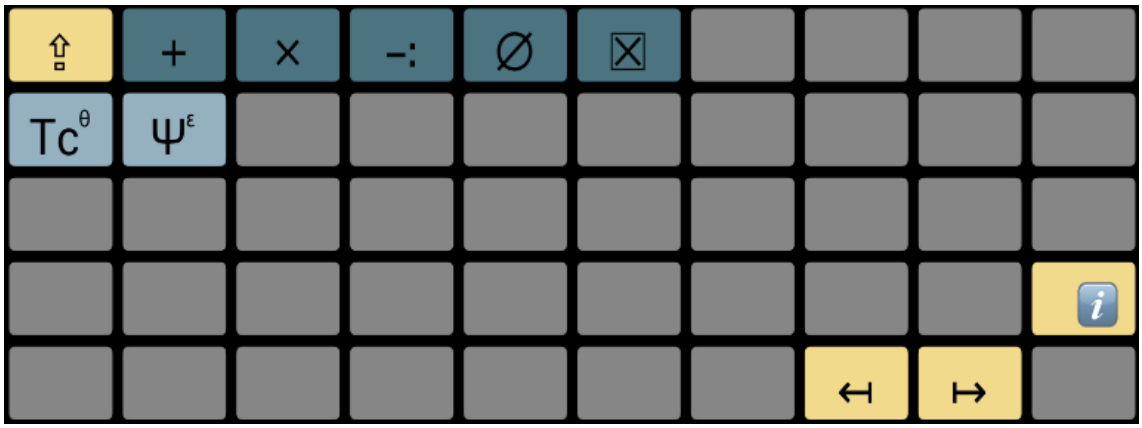
SIMPLE ASSIGNMENT KEYBOARD

This enables the target variable to be associated with a value assignment, or to apply a modifier to it (+, – , × , ÷, ◊). When the writing pointer is placed in front of an assignment instruction, the special assignment keyboard is shown. The first row has the instruction modifiers. The first key is the 'New Line' key (⇪), which enables users to add a new program instruction in front of the assignment. (The 'New Line' key is then displayed.) This is followed by the assignment modifiers, which enable users to add an operator to the assignment that is executed using the result's previous value. By pressing '+', the result of the expression assigned is added to the previous target value. By pressing '–' the result is subtracted, whilst the '×' and '÷' keys are used to multiply or divide it. This makes it possible to create complex calculations in a number of successive assignments by combining the results of previous ones. The next key (ℜ) is for results. It is used to assign the result of an expression to the value of the result (this is the default value). The 'Delete' key (☒) removes the whole line. The rest of the keyboard is filled by the simple values stored in the library. By pressing any one of them, the value is set as the target of the assignment. The keyboard also has an area for names whose keys are the parameters of the formula being defined and the auxiliary values, which can also be used for the calculation.



LIST ASSIGNMENT KEYBOARD

This is used to assign the result of a repeated calculation to the target list, or to make a modification to the way it is assigned (+, − , × , ÷, ◊). When the cursor is placed in front of a list assignment instruction, a special list assignment keyboard is displayed. The instruction modifiers can be found on the first row. As they are identical to those on the simple assignment keyboard with the same functions, they are not discussed here (see the relevant section for full details). The only difference is that there is no ℜ key (for obvious reasons) and that the values shown on the rest of the keyboard to be used as the assignment target are the existing lists.



PROGRAMMING

The startup screen is the starting point for setting the elements that will configure the program. To start, select a category and a type of element. The type and category selected are displayed on the list of elements. By changing the category or type, the list is updated. If elements in a new category are to be created, the category must be created first and then selected. All new elements created are assigned to the category and type selected.

The repository is handled according to the category and type of element selected. By selecting one of the actions from the buttons on the toolbar, the program will go to the right place for it to be executed:

- **New**. This enables an new element of the type selected to be created and it is then added to the category highlighted.

- **Edit**. This modifies the way the element highlighted is programmed.

- **Duplicate**. This enables a new element to be created from an existing one by copying its content and editing it.

- **Remove**. This removes an element that has been highlighted from the library, once the program has checked that it is not being used by any other element.

- **Calculate**. This is used to go to the program execution mode.

Except in the case of ‘Remove’, the program switches to the work screen where the action selected is executed.

In the case of a new element (of the category and type highlighted) and the modification or duplication (of the element selected), the program switches to the editing screen, which has three windows: **Name**, **Content** and **Keyboard**. Each type of element configures these windows differently depending on the action to be performed. The general programming features are described below and then each specific case is discussed.

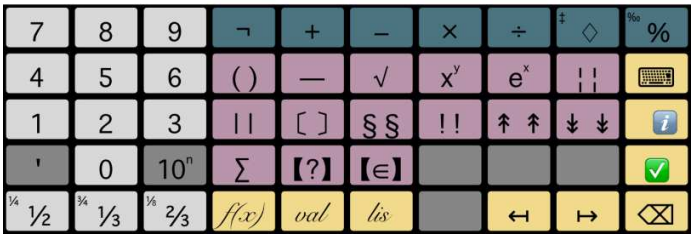
1. **Programming screen.**
2. **Name screen.**
3. **Programming of values.**
4. **Programming of lists.**
5. **Programming of formulae.**
6. **Formulation window.**
7. **Basic operators.**
8. **Groups.**
9. **Integrated functions.**
10. **Lines.**
11. **Browsing.**
12. **Templates.**

PROGRAMMING SCREEN

The second window on **Eureka** is the programming screen. Program elements can be created and calculations made from this screen. It is divided into three:

• **The name or results section.** In order to edit an element (formula, value, list or template), this section serves to create the heading and symbolic name of an element in accordance with the type, the associated configuration values or (for formulae) the call parameters and the auxiliary calculation variables. The results of calculations are displayed here. It is generally divided into three parts.
• **Content box**. The program associated with the type of element being defined is entered here. (This section is disabled in Values).
• **Keyboard**. The keyboard is at the bottom of the screen. Its configuration is automatically changed so that the symbols belonging to the programming phase in which an entry is found are always displayed.

The constant values or general pre-programmed mathematical functions that are integrated in the calculator are on the second page of the keyboard. They go by their generally accepted names, so no further explanations are required. By pressing the key of a constant value or integrated function, a call to its symbol is added and, depending on the format, the parentheses open to show the calculation parameters or the cursor places itself to display the parameter as a subscript.



NAME SCREEN

The first box in the programming window is the 'Name' window. It contains the following information:

• **Label**

This is a descriptive text associated with a given element to define its function. An alphanumeric keyboard is displayed for the headings that can be used to type in whatever text the user wishes to add.

• **Name**.

The name of an element is entered at the bottom, as it will appear in the mathematical expressions in which it is used. A great variety of symbols traditionally used in mathematics are allowed for representing the names of formulae, values and tables. This enables users to assign a direct, intuitive and simple image to each element of calculation. As is the standard practice in mathematical elements, they should be short and simple. Each element can have a name of up to 8 characters. The first is the main one and can have 'embellishments'. The remainder can be set on the same line of type, or as subscripts or superscripts, or both. As is the standard practice in mathematical expressions and scientific notation, it is recommended that the name be based on a letter, which can be Latin or Greek, uppercase or lowercase, or an additional set of auxiliary characters (ornamental, Hebrew, symbols, etc.). An element must comprise one (or two if necessary) main letters, an embellishment added in the case that an element is to be created that is related to the same symbol, and one or two letters or numbers in subscript or superscript in order to create names that are clear, short and easy to handle. The first letter of a name can take an embellishment. An embellishment is a punctuation mark superimposed on the main letter in order to distinguish its function from other similar ones, and this creates a set of related symbols. The same effect can be obtained by using subscript and superscript. A special type of embellishment that is not visible makes it possible to embed the second letter in the first, at three different levels, in order to create the effect that they are superimposed, i.e. [° ˆ ¯ ˜ ’ ″].

Whilst the name window is open and its content is being defined, the alphabetical keyboard is enabled. In order to enable this window from the content window, touch it on the left-hand side. It recovers focus, changes the background colour, configures the keyboard and displays the cursor in the place it should be.

To go from the name to the content (formulation) window, just touch the latter anywhere.

PROGRAMMING OF VALUES

A '**Value**' is understood to be a programming element that assigns a name to a piece of simple numerical content. The values are names assigned to numbers.

A value can be fully defined in the name window without the need to use the content window. As described previously, a descriptive label is written in the name window based on the value, as is the name, which will be used to designate it in the formulae in which it is used.

The following additional information is added:

• **Scale**

This shows the value's unit of measure, which may be:

- o The nominal value that is entered.
- o %. The value is expressed in percentage terms.
- o ‰. The value is expressed as parts per mille.
- o ‱. The value is expressed as parts per million.

• **Def**

The default value, or the numerical value assigned to the variable for use in the formulation.

PROGRAMMING OF LISTS

A '**List**' is understood to be a number of numerical values that can be accessed in the formulation from an access index.

There are two steps to defining a **list**: its name and attributes are set in the **Name** window and then the values are entered in each of the values that make up the list.

As described previously, a descriptive label is written in the name window based on the list, as is the name, which will be used to designate it in the formulae in which it is used.

The following additional information is added:

- **Format**

  The format for displaying an element is set according to its type on the left-hand side of the line. The format enables users to decide how the search index of the numerical element in the list is displayed when it is used in a mathematical expression (formula). The formats allowed for lists are:

  - **[x]** Square brackets.
    The parameters are written without a name (in order of appearance) between square brackets, e.g.: Ψ_5[n].

  - **Sub** Subscript.
    The parameters are seen as unnamed subscripts, e.g.: Ψn

- **Scale**

  This shows the units of measure of the values in a list, which may be:

  - The nominal value that is entered.
  - %. The value is expressed in percentage terms.
  - ‰. The value is expressed as parts per mille.
  - ‱. The value is expressed as parts per million.

- **Def**

  The default value, or the numerical value that the list will return when a given element searched for does not exist.

- **Max**

  This is the number of elements that make up a list and that are created when the value window is opened. If this value is modified, the size of the list changes immediately by inserting the elements (gaps) that are missing or by deleting those that are left over.

• **Index**

This controls the behaviour of a list at the time a search for an element is made. It can be:

- **A sequence.**
  The index is a consecutive number from zero upwards.
- **From.**
  The index forms part of a set of values starting from one given value (inclusive) to another (exclusive).
- **Up to.**
  The index forms part of a set of values starting from the previous value (exclusive) going up to a given value (inclusive).
- **Key.**
  A search is made for an element whose search index exactly coincides with the key shown in each element.

Each of the elements that make up a list are entered into the work window. Each list comprises two columns: index and value. Depending on the type of access index (sequence, from, up to or key), the index runs automatically (sequence) or must be entered together with a value. The up and down arrow keys are used to move from one element to another. The left and right arrow keys place the writing pointer on an element.

PROGRAMMING OF FORMULAE

Eureka's main task is to programme formulae, which will be stored in your personal library for direct use in calculations on a recursive basis.

The first step in setting up a formula is to create a heading to identify it in the '**Name**' window. In addition to the label and name, which follow the standard rules of nomenclature, the following must be defined for each formula:

• **Format**

The format for displaying an element is set according to its type on the left-hand side of the line. The format enables users to decide how the parameters of a function call are displayed when used in a mathematical expression (formula). If a formula has more than one parameter, they are separated by the operator 'Separate' '♦'. The formats allowed for formulae are:

1. **()** Parentheses

   The parameters are written without a name (in order of appearance) between parentheses,

   e.g.: Ψ_5(1♦ 2).

2. **[]** Square brackets

   The parameters are written without a name (in order of appearance) between square brackets,

   e.g.: Ψ_5[1♦ 2].

3. **[n⇦x]** Named

   The parameters are written between square brackets, and each one displays its assigned name and operator,

   e.g.: Ψ_5[α⇦1 ♦ β ⇦2].

4. **Sub** Subscript.

   The parameters are seen as unnamed subscripts,

   e.g.: Ψ 1♦ 2

• **Parameters**

The formulae enable call parameters to be defined that will later be replaced by real values in the expression in which they are used by running the parameter parse call, as is standard in scientific calculation and in programming languages. Up to a maximum of 8 parameters can be set for each formula. Each parameter has a name, which is made up of one to three letters (uppercase, lowercase, Latin or Greek) that can be written as subscript. It is recommended that the parameters be given names that are concise, clear and straightforward. These names are used in the mathematical expressions that make up the formulae that are defined, and during runtime their value is substituted by the call value. When a formula in an expression is called,

these names can be seen in the expression if so specified in the format of the call (second form). In forms one and two, only the position but not the name is taken into account.

- **Local values**

The multiline formulae enable local values to be set that are used as auxiliary variables in a formula in order to store interim results, as occurs in programming languages. On exiting a formula, these values are lost. Up to a maximum of 8 parameters can be set for each formula. Each local value has a name, which is made up of one to three letters (uppercase, lowercase, Latin or Greek) that can be written as subscript. It is recommended that the local values be given names that are concise, clear and straightforward. These names are used in the mathematical expressions that make up the formulae that are defined.

FORMULATION WINDOW.

In the content edit section of a formula (programming window), the set of self-configiring keybpards are used to enter the mathematical expression that makes up the formula.

A formula can be a simple mathematical expression (a single line) of a complex one (multiline).

The variable ℜ receives the result. The result returned by a formula when it is called in an expression by another formula or calculation is the final content of this variable, regardless of whether partial results can be assigned to other value variables in the library.

When a formula is edited, only the left and right arrow keys may be used to move around the text of a formula in order to add or modify an expression. The left arrow key is used move backwards in the code and the right arrow to move forwards until the writing pointer is in the right place. The writing pointer is shown as a red vertical line between two programming symbols. Any new symbols will be inserted exactly between the two symbols where the writing pointer is found (or at the beginning or end of an expression), and all other expressions will be moved accordingly. As expressions are not represented linearly due to the existence of mathematical variables (numerators, denominators, powers, summations), care must be taken whenever a formula is edited and its basic internal structure fully understood.

A formula is an algebraic expression used for calculation that is represented internally as a sequence of symbols arranged as they appear on screen and that is also the order in which they are calculated. This is why it is only possible to go backwards or forwards in the order expressions are arranged, which must be borne in mind at all times. Furthermore, an element can cause confusion until the existence of logical groupings (GROUPS) is assimilated. As discussed previously, a group is a part of an expression that is evaluated on its own. The basic model uses parentheses. An expression between parentheses has its own identity and is evaluated first. All groupings act as parentheses but have calculation directives that give them their own identity. The groupings in Eureka, as well as formulae added and defined by users, will be discussed below.

Eureka always writes the opening and closing parentheses in a single operation in order to make it easy to write expressions and ensure their coherence, and avoiding any errors made in the balancing of parentheses. This has two consequences. Firstly, that whenever a GROUP key is pressed, the program's open and close symbols will always be inserted. In some cases they will be seen and in others they will not. The writing pointer must always be placed between them in order to write the content. The way to exit a GROUP once an expression has been completed is to move the writing pointer using the right arrow key so that it is in front of the close symbol.

If a GROUP is not explicitly visible (i.e. the open and close symbols are implicit), it must be handled in the same way. However, as the open and close symbols cannot be seen it is more difficult and, therefore, it is advisable to have a firm grasp of when the cursor is inside or outside of the GROUP. Fractions, powers, exponentiation, square roots, summation indices and table subscripts are invisible. Parentheses, absolute values, integers, decimal numbers, factorials, floors and ceilings are visible. It is possible in all cases to know where the cursor is from its position, size, colour or effects. It is important to get used to this idea. If a symbol is written inside a GROUP, it is simply deleted and the right arrow key used to exit it.

It is also important to be aware of how to delete GROUPS. As discussed above, the open and close symbols are inserted simultaneously and must never appear unpaired or out of order. This means that if a GROUP is deleted, both the open and close symbols will be removed at the same time. To avoid accidentally deleting whole expressions, Eureka requires that a GROUP must be empty, that is, there must not be any expression inside it, before it can be deleted. A GROUP cannot be deleted if there is anything written inside it. The rules are: never directly ?delete a close symbol. If an attempt is made to do so, the command will simply not be executed. If the delete key is pressed behind a GROUP close symbol, nothing will happen. In the case of invisible GROUPS, it should be taken into account that the cursor does not go backwards. Secondly, before a GROUP can be deleted the expression inside it must be removed. Once it is empty, it can be deleted by pressing the delete key at the beginning of the GROUP (open). For GROUPS with separators (maximum, minimum, functions with more than one parameter) only the separators should be left inside them.

The various elements that make it possible to create mathematical expressions in a formula are discussed below.

BASIC OPERATORS

Basic operators are a set of elemental symbols that carry out basic arithmetic operations.

¬ Change of sign. This changes the sign of the following value or expression.

+ Add. This adds up the previous and following values or expressions.

– Subtract. This subtracts the difference between the previous and following values or expressions.

× Multiply. This multiplies the previous and following values or expressions,

÷ Divide. This divides the previous value or expression by the following one. This is an alternative to the fraction operator (GROUP).

◊ Modulus. This calculates the modulus, or the rest of a long division, between the previous and following values or expressions.

‡ Integer quotient. This divides values into each other and gives an integer quotient as a result.

% Percent. This operator enables percentages to be calculated, depending on the previous operator. If the previous operator is '+', the first value will be increased by X%; if it is `-`, it will take away X%. If it is `×`, the % value is calculated; and if it is '÷' the result obtained is the % value of the first value over the second.

50 + 4 % ⇨52

50 – 4 % ⇨48

50 × 4 % ⇨ 2

45 ÷ 150 ⇨30 %

‰ Per mille. This works in a similar way to % but in parts per thousand.

GROUPS

Groups, or grouping operators, are made up of two symmetrical symbols (open and close) that enclose or group together an expression on which the action associated with a group's operator is executed. Some open and close symbols can be seen, but are displayed by a change in size and position (subscripts and superscripts) in line with standard mathematical notation.

() Parenthesis

This serves to determine the order of precedence of calculations, in the standard way used in mathematical calculation.

— Fraction

A fraction bar opens that enables a numerator and a denominator to be entered. In order to go from the numerator to the denominator the cursor must be moved using the right arrow key, which is also used to exit a fraction.

x^n Power

This opens a group in which the value of an exponent can be stated. To exit the exponent, the cursor must be moved using the right arrow key.

e^n Exponentiation

This opens a group in which the value of the exponent can be raised to the power 'e' (base of Napierian logarithms). To exit the exponent, the cursor must be moved using the right arrow key.

√ Square root

This opens a group in which the value on which the square root is calculated can be stated. Its length is worked out from the size of the top line. To exit this function, the cursor must be moved using the right arrow key.

!! Factorial

This calculates the factorial of an expression or its included value.

|| Absolute value

This removes the sign of the expression entered.

⁞⁞ Integer

This extracts the integral part from the result of an expression and disregards decimal part.

§§ Decimal part

This extracts decimal part from the result of an expression and disregards the integral part.

↥↧ Ceiling

This returns the next highest integer by rounding up the result.

↧↥ Floor

This returns the next lowest integer by rounding down the result.

⇑⇑ Maximum

This returns the highest value on a list of values or expressions.

↡↡ Minimum

This returns the lowest value on a list of values or expressions.

[?] Decision table with conditions

Three values or expressions are entered here. The first, in front of the '?' sign, handles an operation. If the result is true, it returns the second expression as a value and the third if it is false. The separator symbol '⌥' is used to separate the second and third expressions.

[5 > 3 ? 18 ⌥ 35] ⇨ 18

[∈] Indexed decision table

An initial expression must be entered, which will be the search key. A list of expressions is then entered that is made up of two parts: key and value. If the search key is in the key of the element, it is returned as the result of the value. Finally, a default value can be set that will be returned if the key has not been found in any element. This complex operation uses the following symbols:

▷ This separates the search key from the rest.

⌥ This separates the various search elements. Each element comprises a key and results field.

⇉ This separates the key of each search element from the result. What appears on the right is the value that is returned if the search key is in the element's key field.

¨ This separates the various simple values in the key field from the element.

↔ This creates key value intervals. The interval ranges from the value in front of the symbol to that behind it.

‣ Default value. This is the value that is returned if the search is unsuccessful.

For example: [index ▷ 5 ♦8 ♦12 ↔22 ⇉2'34 ⌥30 ♦31 ⇉7'12 ‣8'56], which means that if the value of the '**index**' is 5 or 8, or is in the range of 12 to 22, the result is 2'34. If the value of the '**index**' is 30 or 31, the result is 7'12. In all other cases, the result will be 8'56.

INTEGRATED FUNCTIONS

The constant values or general pre-programmed mathematical functions that are integrated in the calculator are on the second page of the keyboard. They go by their generally accepted names, so no further explanations are required. By pressing the key of a constant value or integrated function, a call to its symbol is added and, depending on the format, the parentheses open to show the calculation parameters or the cursor places itself in the right place to display the parameter as a subscript.

There are three different blocks:

1. Advanced math functions**.**

   π. Mathematical constant π (ratio of the circumference to its diameter).

   e. Mathematical constant e (base of natural logarithms).

   log. Natural logarithm (base 10).

   ln. Natural logarithm (base e**).**

   Trigonometric. sin (sine), cos (cosine), tan (tangent).

   Inverse trigonometric. asin (arc sine), acos (arc cosine), atan (arc tangent).

   Hyperbolic trigonometric. Represented by an 'h' in the corner of each key, called the hyperbolic sine, hyperbolic cosine, hyperbolic tangent respectively.

2. **Funciones estadísticas y de tablas.** Estas funciones se aplican a una lista definida previamente de forma global y simultanea. La lista se indica a continuación del operador de lista '⇨'.

   C_n⇨T_b. Devuelve el número de elementos de la lista (Count).

   S_n⇨T_b. Devuelve la suma algebraica de los elementos de la lista (Sum).

   A_v⇨T_b. Devuelve la media aritmética simple de los elementos de la lista (Average).

   C_n⇨T_b. Devuelve la media aritmética de los valores de la lista ponderados con los de su clave.

   C_n⇨T_b.

   C_n⇨T_b.

   C_n⇨T_b.

3. **Funciones de cálculo actuarial.**

LINES

Although most calculations can be expressed in a single expression, which evaluates a formula and assigns it to the result, one of **Eureka's** essential programming features is that a calculation procedure (formula) can contain various lines, which together evaluate the final result. Each line is identified by the operator that controls it, which can either be an assignment operator (evaluates an arithmetical expression and assigns its result) or a decision operator. Depending on the control expression, **Eureka** decides which path to take.

a. **Simple assignment**

This assigns the result of an expression to the assignment variable. By creating an assignment (⇦), the variable 'Result' (ℜ) is set as the receiver variable by default. It contains the final result that a formula will return or the result of the calculation being made. By placing the cursor at the beginning of an expression, the 'assignment keyboard' displays the possible modifications that can be made. This keyboard can be used to change a calculation's receiver variable by setting any global variable stated in the general 'Values' tab, or any local variable or parameter stated in the header of a formula ('Name' window). The 'Calculate' function automatically defines 8

predetermined local variables, which are marked as up to . It is also possible to add calculation modifiers, which affect the assignment. As they are arithmetic operators (+, -, ×, ÷), the assignment variable is used to operate (by adding, subtracting, multiplying or dividing), rather than the result being directly assigned to the assignment variable.

b. **Table assignment**

This generally works in the same way as a simple assignment, but the result is assigned to (or used to operate) each of the elements in the assignment table to create a calculation loop, in which the calculation of an expression is repeated for each of the elements in the table, thus giving a list of results instead of a single result. In order to control a loop, a local variable 't' is automatically created that appears in the basic keyboard and can be used in calculations.

c. **Decision**

This is the ability to create a program block (of one or several lines) according to the result of the 'logical' expression that serves as the control. A logical expression may either be true or false. Whenever a logical expression is being written, comparison and Boolean operators appear in the corner of the keyboard. (<, >, =, ≤, ≥, ≠,∈). The decision block is made up of a 'true' set (✓), which is evaluated whenever the control expression gives a true result, and a 'false' set (×), which is evaluated whenever the control expression gives a false result.

d. **Selection**

This is the ability to create a different program block (of one or several lines) depending on the result of the control expression. Once an expression has been evaluated, its result is compared with each of the selection values and the code of the block belonging to the coincident selection is executed. Otherwise, there is a default selector that evaluates a block of code should no coincidence be found.

e. **Repetition**

This consists in evaluating the block of instructions (of one or several lines) repeatedly, depending on the end-of-test-criterion. There are three options:

1. Repetition of a condition
   The block is repeated provided the result of the logical control expression is fulfilled (it gives true as a result). A logical expression may either be true or false. Whenever a logical expression is being written, comparison and Boolean operators appear in the corner of the keyboard (<, >, =, ≤, ≥, ≠,∈).
2. Indexed repetition
   A block is repeated as many times as there are elements in the control interval, which consists of a series of intervals (initial and final values) or set values. The local variable takes the value of the index every time it is executed.
3. Repetition of a list
   A block is repeated once with each of the values in a list. As the list in an expression is used, it is not necessary to establish an index as it is assumed that it is the element currently being evaluated.

f. **Configuration**

The configuration functions do not actually make calculations and, therefore, they do not form part of the algebraic expressions calculator. Their purpose is to establish general calculation values and the overall framework in values or units. They are:

1. ConfMort [Life table#interest#period#sex#excess mortality]
2. ConfInt [actual annual rate of interest#calculation period]

The configuration functions store the values, as well as calculating the set values for any one period (monthly, two-monthly, three-monthly, four-monthly, six-monthly, annually), the linear interpolation life tables, the actual rate of interest in a period, and the values 'u' and 'v'.

BROWSING

When a formula is edited, only the left and right arrow keys may be used to move around the text of a formula in order to add or modify an expression. The left arrow key is used to move backwards in the code and the right arrow to move forwards until the cursor is in the right place. The cursor is shown as a red vertical line between two programming symbols. Any new symbols will be inserted exactly between the two symbols where the cursor is found (or at the beginning or end of an expression), and all other expressions will be moved accordingly. As expressions are not represented linearly due to the existence of mathematical variables (numerators, denominators, powers, summations), care must be taken whenever a formula is edited and its basic internal structure fully understood.

A formula is an algebraic expression for making linear calculations, which are arranged in the order their components appear on screen and are calculated accordingly. This is why it is only possible to go backwards or forwards in the order expressions are arranged, which must be borne in mind at all times. Furthermore, an element can cause confusion until the existence of logical groupings (GROUPS) is assimilated. As discussed previously, a group is a part of an expression that is evaluated on its own. The basic model uses parentheses. An expression between parentheses has its own identity and is evaluated first. All groupings act as parentheses but have calculation matrices that give them their own identity. The groupings in Eureka, as well as formulae added and defined by users, have already been discussed above.

Eureka always writes the opening and closing parentheses in a single operation in order to make it easy to write expressions and ensure their coherence, and it corrects any errors made in the balancing of parentheses. This has two consequences. Firstly, that whenever a GROUP key is pressed the program's open and close symbols will always be inserted. In some cases they will be seen and in others they will not. The cursor must always be placed between them in order to write the content. The way to exit a GROUP once an expression has been completed is to move the cursor using the right arrow key so that it is in front of the close symbol.

If a GROUP is not explicitly visible (i.e. the open and close symbols are implicit), it must be handled in the same way. However, as the open and close symbols cannot be seen it is more difficult and, therefore, it is advisable to have a firm grasp of when the cursor is inside or outside of the GROUP. Fractions, powers, exponents, square roots, summation indices and table subscripts are invisible. It is possible in all cases to know where the cursor is from its position, size, colour or effects. It is important to get used to this idea. If a symbol is written inside a GROUP, it is simply deleted and the right arrow key used to exit it.

It is also important to be aware of how to delete GROUPS. As discussed above, the open and close symbols are inserted simultaneously and must never appear unpaired or out of order. This means that if a GROUP is deleted, both the open and close symbols will be removed at the same time. To avoid accidentally deleting whole expressions, Eureka requires that a GROUP must be empty, that is, there must not be any expression inside it, before it can be deleted. A GROUP cannot be deleted if there is anything written inside it. The rule is to never directly delete a close symbol. If an attempt is made to do so, the command will simply not be executed. If the delete key is pressed behind a GROUP close symbol, nothing will happen. In the case of invisible GROUPS, it should be taken into account that the cursor does not go backwards. Secondly, before a GROUP can be deleted the expression inside it must be removed. Once it is empty, it can be deleted by pressing the delete key

at the beginning of the GROUP (open). For GROUPS with separators (maximum, minimum, functions with more than one parameter) only the separators should be left inside them.

TEMPLATES

What makes Eureka a veritable mathematical programming language is the addition of the template module. A template is a user-friendly form on screen that enables users to interactively input and output data and see them clearly displayed.

A work method has been devised that sidesteps the complications inherent to handling input and output fields, whereby their position, size and aesthetic features are defined. To make the work of users easier, a simple environment has been created that automates most tasks by applying pre-set criteria, as well as displaying information so that it is easy to see on screen.

There are two steps to creating an interactive template. The first is to design the template form, which consists in defining all of the input and output fields that make it up. The second consists in applying the formulation and calculation to each of the output fields defined using the values of the input fields. The fields are automatically arranged on screen through a mechanism that places them in containers, which set their position.

A field consists of a label that describes it and the field name (which follows the same rules for naming all the other programming elements). An alternative option is to add a queue, which is a text that is displayed behind the field (to show units or additional information).

The fields are divided into panels. A panel is a space on the screen where the input and output fields are linearly arranged in succession. Its sole purpose is to arrange fields and decide where they should be placed on screen. There are horizontal and vertical panels. This is their main property. On a vertical panel, the fields that make it up are displayed vertically on top of each other. The field label appears first (on the left) and the field is displayed below it to form a column. On a horizontal panel, the fields are displayed in a single line one after the other. The field label appears just above the field in the shape of a header. Fields cannot be created outside a panel.

The panels are likewise placed in frames. A frame is simply a container of panels. A frame can contain as many panels as are necessary, within the constraints of the size of the screen. A frame can be horizontal or vertical. A horizontal frame places its panels next to each other, all of which are assigned the same space (width and height). A vertical frame places its panels on top of each other.

The system therefore has a hierarchical structure with all of the levels necessary to obtain the format required. The first element will always be a frame (horizontal or vertical). As many other frames or field panels as required can be added.

The templates are designed in such a way that their various components (frames, panels and fields) are each defined on a separate programming line.

The programming window is divided into two parts: the type and features of an element that is being defined are entered in the left-hand column, and the label, name and optional queue, in the case of fields, are entered on the right-hand side in free format. In each case, the virtual input key adjusts to the needs of what is being defined.

If the cursor is placed on the left-hand column, the element is shown in a red box and the type of element keyboard appears. The type of element to be defined (MarcoHor, MarcoVer, Panel Hor, Panel Ver, Campo) and its features can be set. Any elements that do not fulfil the set rules of hierarchy cannot be defined.

The second part of the process in creating a template is to assign calculations to the output fields. Using the 'switch' key (...), the status is switched. The formula programming window is displayed using its keyboard. The only difference with the procedure that is normally used to programme formulae is that when a keyboard with simple values is called, in the case of both calculation and assignment the fields defined are also displayed, as a result of which both the input fields can be used as a calculation value and the output fields as a simple assignment field.

Once a template has been created, it is saved like all other programming elements and forms part of the repository. To use it, select the initial access screen and click on the 'calculation' button.