EasyBite Programming Language

EasyBite is a beginner-friendly programming language designed to introduce kids and beginners to the world of programming. It aims to provide a simple syntax and easy-to-understand concepts while still offering essential programming functionalities.

Table of Contents

- getenable(formName: string, controlName: string): bool
- getforecolor(formName: string, controlName: string)
- getimage(formName: string, pictureBoxName: string)
- getitem(listName: string, index: number): any
- gettext(formName: string, controlName: string)
- getmin(formName: string, controlName: string): int
- getmax(formName: string, controlName: string): number
- getstyle(formName, controlName)
- getvalue(formName: string, controlName: string): any
- getvisible(formName: string, controlName: string) : bool
- getx(formName: string, controlName: string): int
- gety(formName: string, controlName: string): int
- groupbox(formName: string, [text: string], [left: int], [top: int])
- hideform(formName: string)
- label(formName: string, text: string, [fontName: string], [fontSize: int], [fontStyle: string], [foreColor: string], [backColor: string], [top: int], [left: int])
- listbox(formName: string, [labelText: string], [top: int], [left: int], [width: int], [height: int])
- messagebox(formName: string, title: string, message: string, [buttons: string], [icon: string], [defaultButton: string])
- panel(formName: string, [left: int], [top: int])
- picturebox(formName: string, [names: string], [imagePath: string], [width: int], [height: int], [top: int], [left: int])
- progressbar(formName: string, [names: string], [minimum: int], [maximum: int], [value: int], [width: int], [height: int], [top: int], [left: int])
- radiobutton(formName: string, [controlName: string], [text: string], [isChecked: bool], [x: int], [y: int])
- runapp(appName: string)
- setabove(formName: string, targetControlName: string, controlName: string, [spacing: int])
- setalignment(formName: string, controlName: string, alignment: string)
- setbackcolor(formName: string, controlName: string, color: string)
- setbelow(formName: string, targetControlName: string, controlName: string, [spacing: int])
- setdock(formName: string, controlName: string, dockStyle: string)
- setenable(formName: string, controlName: string, enable: bool)
- setforecolor(formName: string, controlName: string, color: string)
- setimage(formName: string, pictureBoxName: string, imagePath: string)
- setitem(listName: string, index: number, value: any)
- setleft(formName: string, targetControlName: string, controlName: string, [spacing: int])
- setright(formName: string, targetControlName: string, controlName: string, [spacing: int])

Installation

To use EasyBite, you need to have the EasyBite interpreter installed on your machine. Follow the steps below to install it:

1. Download the EasyBite interpreter from the official website or download it here in github at the top.
2. Run the installer and follow the installation instructions.
3. During the installation process, the installer will check if .NET Framework 4.6 or later is already installed on your machine. If it is detected, the installation will proceed to the next step. If it is not detected, you will be prompted with two options:
    - Option 1: Download and install .NET Framework from the official Microsoft website. Choose this option if you prefer to download and install .NET Framework separately. Follow the instructions provided by the installer to complete the installation.
    - Option 2: Install .NET Framework from the EasyBite setup. Choose this option if you want to install EasyBite along with the required .NET Framework version in one step. The EasyBite setup will automatically install .NET Framework for you.
4. Once the installation is complete, you can use the EasyBite interpreter from the command line.
5. To run an EasyBite file, open the command prompt or terminal and navigate to the directory where your EasyBite file is located.
6. Type `EasyBite yourfilename.bite` and press Enter to execute the EasyBite program.

If you encounter any issues during the installation process or have any questions, please don't hesitate to contact us at muhammadgoni51@gmail.com. We are here to assist you.

Syntax

EasyBite has a simple and intuitive syntax that makes it easy to write and understand code. Here are some key elements of the EasyBite syntax:

Syntax Highlight

As EasyBite is a relatively new language, it currently does not have native support for syntax highlighting. However, users can follow a process to manually add indentation and syntax highlighting in Sublime Text. While we are actively working on developing an integrated development environment (IDE) for EasyBite, we kindly ask users to be patient and wait for the upcoming release.

In the meantime, we provide the following steps for configuring indentation and syntax highlighting in Sublime Text for EasyBite:

1. Open Sublime Text and go to "Preferences" in the top menu.
2. Select "Browse Packages" from the dropdown menu. This will open the Packages folder in your file explorer.
3. Create a new folder inside the Packages folder and name it "EasyBite" (or any name you prefer).
4. Inside the "EasyBite" folder, download the "EasyBite.sublime-syntax" and "EasyBite.tmPreferences" files from the EasyBite GitHub repository.
5. Move the downloaded files into the "EasyBite" folder.

Next, we'll configure the indentation rules for EasyBite:

1. Open Sublime Text and go to "Preferences" in the top menu.
2. Select "Settings" from the dropdown menu. This will open the Sublime Text settings file.
3. Add the following line to the settings file: `"translate_tabs_to_spaces": true`. This ensures that indentation is based on spaces instead of tabs.
4. Save the settings file.

After completing these steps, Sublime Text should recognize and apply the indentation and syntax highlighting rules for EasyBite files. When you open a file with the ".bite" extension or explicitly set the syntax to EasyBite, Sublime Text will provide proper indentation and highlight the syntax elements according to the defined rules.

Please note that these manual configurations serve as a temporary solution until the official EasyBite IDE is released. We appreciate your patience and understanding as we work towards providing a more streamlined and user-friendly development environment for EasyBite.

Comments

Use the double forward-slash (`//`) to add single-line comments in your code.

```
// This is a comment
```

Note: Multiline comment is not yet supported.

Keywords

Keywords, also known as reserved words, are words that have a special meaning and purpose in a programming language. These words are predefined by the language and cannot be used as identifiers (such as variable names, function names, etc.) because they are reserved for specific purposes within the language's syntax and grammar.

Keyword	**Keyword**	**Keyword**	**Keyword**	**Keyword**
declare	set	show	if	then
else	end if	for	from	to
step	generate	by	stop	repeat
while	times	iterate	over	choose
when	otherwise	true	false	function
return	end function			

Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**
class	new	method	inheritance	secret
public				

Keyword	**Description**
`declare`	Used for variable declaration
`set`	Assigns a value to a variable
`show`	Prints the value of a variable or expression
`if`	Begins an if statement
`then`	Indicates the beginning of the if block
`else`	Indicates the beginning of the else block
`end if`	Ends an if statement
`for`	Begins a for loop
`from`	Specifies the start value of the for loop
`to`	Specifies the end value of the for loop
`step`	Specifies the step size for the for loop
`generate`	Begins a generate loop
`by`	Specifies the step size for the generate loop
`stop`	Stops the execution of a generate loop

Keyword	Description
`repeat`	Begins a repeat loop
`while`	Specifies the condition for the repeat loop
`times`	Specifies the number of times to repeat
`iterate`	Begins an iteration over an array
`over`	Specifies the array to iterate over
`choose`	Begins a choose statement
`when`	Specifies a condition in a choose statement
`otherwise`	Specifies the default case in a choose statement
`true`	Represents the boolean value true
`false`	Represents the boolean value false
`function`	Declares a function
`return`	Specifies the return value in a function
`end function`	Ends a function definition

Operators

Operator	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`^`	Exponentiation
`remind`	Modulo (remainder of division)
`and`	Logical AND
`or`	Logical OR
`not`	Logical NOT
`==`	Equal to
`>`	Greater than
`<`	Less than
`<=`	Less than or equal to
`>=`	Greater than or equal to

Operator	Description
!=	Not equal to

Examples

Arithmetic Operators:

```
set a to 10
set b to 5

declare result

set result to a + b   // Addition
show result           // Output: 15

set result to a - b   // Subtraction
show result           // Output: 5

set result to a * b   // Multiplication
show result           // Output: 50

set result to a / b   // Division
show result           // Output: 2

set result to a ^ b   // Exponentiation
show result           // Output: 100000

set result to a remind b   // Modulo (remainder of division)
show result           // Output: 0
```

Logical Operators:

```
set x to true
set y to false

declare result

set result to x and y   // Logical AND
show result            // Output: false

set result to x or y    // Logical OR
show result            // Output: true

set result to not x     // Logical NOT
show result            // Output: false
```

Comparison Operators:

```
set a to 10
set b to 5

declare result

set result to a == b   // Equal to
show result            // Output: false

set result to a > b    // Greater than
show result            // Output: true

set result to a < b    // Less than
show result            // Output: false

set result to a <= b   // Less than or equal to
show result            // Output: false

set result to a >= b   // Greater than or equal to
show result            // Output: true
```

Inputs:

```
set name to input("Enter your name: ")
show "Hello, " + name
```

Output:

Enter your name: Goni

Hello, Goni

In the example above, the `input()` function is used to prompt the user to enter their `name`. The value entered by the user is stored in the variable name, and then it is displayed using the show statement.

You can use the `input()` function to get user input for various purposes and perform operations based on that input.

Please note that the input and output in EasyBite may depend on the environment or platform where the code is executed. The above example demonstrates the basic usage of input and output in EasyBite, but the exact behavior may vary depending on the implementation.

Print Statement (`show`)

The `show` statement is used in EasyBite to print the value of an expression to the console. It provides a convenient way to display output during program execution.

Syntax

The `show` statement can be written in two forms:

1. `show expression;`
2. `show(expression);`

The `expression` can be any valid expression in the EasyBite language.

Usage Examples

Here are some examples of how the `show` statement can be used:

Example 1:

```
show "Hello, World!";
```

This statement prints the string "Hello, World!" to the console.

Example 2:

```
show(5 + 3 * 2);
```

This statement evaluates the expression `5 + 3 * 2` (which is equal to 11) and prints the result (11) to the console.

Example 3:

```
set x to "EasyBite";
show("Welcome to " + x);
```

In this example, a variable `x` is defined with the value "EasyBite". The `show` statement concatenates the string "Welcome to " with the value of `x` and prints the result ("Welcome to EasyBite") to the console.

Notes

- The `show` statement is a convenient way to display output during program execution.
- The parentheses around the expression in the second form (`show(expression);`) are optional. The statement can also be written as `show expression;`.
- The `show` statement can be used with any valid expression in EasyBite, including variables, literals, and complex expressions involving arithmetic operations, string concatenation, function calls, etc.

Remember that this documentation assumes familiarity with the EasyBite language and its syntax. If you have any further questions or need additional assistance, please don't hesitate to ask!

If Statement

The `if` statement is used to conditionally execute a block of code based on a specific condition. The condition is specified after the `if` keyword. If the condition evaluates to true, the code block within the `then` and `end if` keywords is executed.

Example:

```
set x to 10

if x > 0 then
    show "x is positive"
end if
```

If-Else Statement:

The `if-else` statement allows for executing different code blocks based on the condition. If the initial condition specified after the `if` keyword is true, the code block within the `then` and `else` keywords is executed. If the condition is false, the code block within the `else` and `end if` keywords is executed.

Example:

```
set x to -5

if x > 0 then
    show "x is positive"
else
    show "x is non-positive"
end if
```

If-Else If Statement:

The `if-else if` statement allows for multiple conditions to be checked sequentially. If the initial condition is false, subsequent conditions can be checked using the `else if` keywords. The first condition that evaluates to true will execute its corresponding code block, and the remaining conditions are skipped.

Example:

```
set x to 0

if x > 0 then
    show "x is positive"
else if x < 0 then
    show "x is negative"
else
    show "x is zero"
end if
```

If Statement with Parentheses:

Parentheses can be used to group the conditions in the if statement, improving readability and ensuring the desired order of evaluation.

Example:

```
set x to 7

if (x > 0) then
    show "x is positive"
end if
```

Loop

For Loop

The `for` loop is used to iterate over a range of values. It consists of three parts: the loop variable declaration, the loop condition, and the loop increment. The loop variable is declared using the `for` keyword, followed by the loop variable name, the keyword `from`, the start value, the keyword `to`, and the end value. Within the loop block, you can perform the desired operations.

Example:

```
for i from 1 to 5
    show i
end for
```

For Loop with Step:

The `for` loop can also include a step value to specify the increment or decrement between each iteration. After the end value, you can use the keyword `step` followed by the step value.

Example:

```
for i from 1 to 10 step 2
    show i
end for
```

Generate Loop:

The `generate` loop is similar to the `for` loop but has a different syntax. It is primarily used for generating a sequence of values. You can specify the start value, the end value, and an optional step value using the `generate` keyword, followed by the loop variable name and the respective values. The loop is terminated using the `stop` keyword.

Example:

```
generate i from 1 to 5
    show i
```

```
stop
```

Generate Loop with Step:

The `generate` loop can also include a step value to specify the increment or decrement between each iteration. After the end value, you can use the keyword `by` followed by the step value.

Example:

```
generate i from 1 to 10 by 2
    show i
stop
```

Repeat While Loop:

The `repeat while` loop is used to repeatedly execute a block of code while a specific condition is true. The condition is checked at the beginning of each iteration. If the condition evaluates to true, the code block within the loop is executed. The loop continues until the condition becomes false.

Example:

```
set x to 1

repeat while (x <= 5)
    show x
    set x to x + 1
end repeat
```

Repeat Times Loop:

The `repeat times` loop is used to execute a block of code a specific number of times. You provide the number of times you want the loop to repeat using the `times` keyword. The loop variable is automatically generated and takes the values from 1 to the specified number of times.

Example:

```
repeat 5 times
    show "Hello, World!"
end repeat
```

Iterate Over Statement

The `iterate over` statement is used to iterate over the elements of an array. It allows you to perform operations on each element of the array within a loop. You specify the loop variable using the `iterate`

keyword, followed by the loop variable name, the keyword `over`, and the array name. Within the loop block, you can access and manipulate the loop variable.

Example:

```
set arr to [1, 2, 3, 4, 5]

iterate n over (arr)
    show n
end iterate
```

Choose Statement:

The `choose` statement in EasyBite is used for conditional branching. It allows you to choose among multiple conditions and execute the block of code associated with the first matching condition. The `choose` statement consists of several `when` blocks and an optional `otherwise` block.

Example:

```
set expression to "cond2"

choose(expression)
    when "cond1":
        show "Condition 1 is true."
    when "cond2":
        show "Condition 2 is true."
    otherwise:
        show "None of the conditions are true."
end choose
```

Function

The `function` statement in EasyBite is used to define a reusable block of code that can be called and executed multiple times with different inputs. It allows you to encapsulate a set of operations into a named function, making your code modular and easier to maintain. The `function` statement is followed by the function name and a comma-separated list of parameters in parentheses. The function block contains the code to be executed when the function is called.

Example:

```
function multiply(a, b)
    return a * b
end function

set result to multiply(5, 3)
show result
```

Recursion is not currently supported in EasyBite. Recursion refers to the process of a function calling itself within its own definition. While recursion can be a powerful technique for solving certain problems, it is not implemented in EasyBite at this time.

If you need to implement recursive algorithms or functions, you may need to explore alternative approaches or adapt your code to iterative solutions. EasyBite focuses on simplicity and ease of use, especially for beginners and young learners, and currently does not include built-in support for recursion.

Please keep in mind this limitation when designing your programs in EasyBite.

Example

```
function factorial(n)
    declare result
    set result to 1

    repeat while(n > 0)
        set result to result * n
        set n to n - 1
    end repeat

    return result
end function

set number to 5
set factorialResult to factorial(number)

show "The factorial of " + number + " is: " + factorialResult
```

In this example, we define a factorial function that takes a number n as input. Inside the function, we declare a variable result and initialize it to 1. Then, we use a repeat while loop to iterate from n down to 1. In each iteration, we multiply the result by the current value of n and decrement n by 1. Finally, we return the computed result.

We then set a variable number to 5 and call the factorial function with number as the argument. The result is stored in the factorialResult variable. Finally, we use the show statement to print the factorial result to the console.

This example demonstrates how to calculate the factorial of a number using iteration instead of recursion in EasyBite.

Arrays

In EasyBite, an array is an ordered collection of elements of the same type. Arrays allow you to store multiple values under a single variable name, making it easier to work with groups of related data.

Example:

```
declare numbers[5]  // Declare an array with a length of 5

numbers[0] to 10
numbers[1] to 20
numbers[2] to 30
numbers[3] to 40
numbers[4] to 50

show numbers[2]  // Output: 30

// Or you can simply do

set myArray to [2, 3, 45, 76, 88]
show myArray[4]
```

Arrays in EasyBite are zero-based, meaning the first element is accessed using the index 0, the second element with the index 1, and so on. You can declare an array with a specific length using the square brackets notation, as shown in the example above.

You can assign values to array elements using the index notation (`array[index] to value`), and you can retrieve the value of an array element by accessing it using the index (`array[index]`).

Arrays in EasyBite can store values of any data type, including numbers, strings, booleans, or even other arrays.

Dictionary (Key-Value Pair)

In EasyBite, a dictionary is a collection of key-value pairs. Dictionaries allow you to store and retrieve values based on their associated keys, making it convenient to work with data that has a unique identifier or label.

Example:

```
set person to {"name": "Goni", "age": 25, "city": "New York"}

show person["name"]  // Output: Goni
show person["age"]   // Output: 25
```

In the example above, we create a dictionary `person` with three key-value pairs: "name" mapped to the value "Goni", "age" mapped to the value 25, and "city" mapped to the value "New York".

You can access the values in a dictionary by specifying the corresponding key in square brackets (`dictionary[key]`). This allows you to retrieve the value associated with a specific key.

Dictionaries in EasyBite are flexible and can store values of different data types as their values. Keys within a dictionary must be unique, but the values can be of any data type, including numbers, strings, booleans, or even other dictionaries.

Import Statement

The `import` statement in EasyBite is used to include external files or built-in libraries in your program. It allows you to access functions, variables, or classes defined in those files or libraries, extending the functionality of your program.

Syntax:

```
import filename
import "library"
from filename import functionname
```

To import a specific file, you can use the import statement followed by the name of the file without the file extension. For example, import utils would import the file named utils.eb.

To import a built-in library, you can use the import statement followed by the name of the library enclosed in double quotes. For example, import "math" would import the built-in math library.

If you only need to import a specific function from a file, you can use the from keyword followed by the filename and the name of the function to import.

Once a file or library is imported, you can access its functions, variables, or classes using dot notation (filename.functionname) or (library.functionname) for built-in libraries.

Import File or Module Location:

In EasyBite, when using the `import` statement to include a file or module, it is expected to be located inside a folder called "modules" in the root directory of your project. This folder acts as a central location for storing reusable code files or modules that can be imported into your EasyBite programs.

The folder structure would look like this:

```
project/
├─ modules/
│ ├─ utils.eb
│ ├─ math.eb
│ └─ ...
├─ main.eb
└─ ...
```

To import a file or module, you can use the `import` statement followed by the filename without the file extension. EasyBite will look for the specified file or module inside the "modules" folder.

For example, to import the file named `utils.eb`, you would use `import utils`. EasyBite will search for the file `utils.eb` inside the "modules" folder.

Ensure that you organize your files and modules within the "modules" folder according to their purpose or functionality, making it easier to manage and locate the desired files when importing.

In-built Libraries and Functions

EasyBite provides several in-built libraries that offer a wide range of functions to simplify programming tasks. These libraries cover various areas such as mathematical calculations, string manipulation, array operations, date and time handling, file operations, and more. Each library has its own set of functions that can be directly used in your EasyBite programs.

To make it easier to navigate through the available libraries and their functions, here is a categorized list:

Math Library

The Math library provides mathematical functions for common calculations. Functions in this library can be used directly without the need for the `Math.` prefix.

Function	Description
`abs(x)`	Returns the absolute value of `x`.
`pow(x, y)`	Returns `x` raised to the power of `y`.
`sqrt(x)`	Returns the square root of `x`.
`sin(x)`	Returns the sine of `x`.
`cos(x)`	Returns the cosine of `x`.
`tan(x)`	Returns the tangent of `x`.
`round(x)`	Rounds `x` to the nearest integer.
`random()`	Returns a random floating-point number between 0 (inclusive) and 1 (exclusive).
`random(start, end)`	Returns a random integer between `start` (inclusive) and `end` (exclusive) if both are provided.
`max(x, y, ...)`	Returns the maximum value among the given arguments.
`min(x, y, ...)`	Returns the minimum value among the given arguments.
`sum(arr)`	Returns the sum of all elements in the given array.
`ceiling(x)`	Returns the smallest integer greater than or equal to `x`.
`floor(x)`	Returns the largest integer less than or equal to `x`.
`log10(x)`	Returns the base 10 logarithm of `x`.
`average(arr)`	Returns the average of all elements in the given array.
`log(x)`	Returns the natural logarithm (base e) of `x`.
`exp(x)`	Returns e raised to the power of `x`.
`mean(arr)`	Returns the mean of all elements in the given array.
`mode(arr)`	Returns the mode (most frequent value) of the given array.
`sign(x)`	Returns the sign of `x` (-1 for negative, 0 for zero, 1 for positive).
`log2(x)`	Returns the base 2 logarithm of `x`.

Function	Description
`sign(x)`	Returns the sign of `x` (-1 for negative, 0 for zero, 1 for positive).

These functions allow you to perform various mathematical operations and calculations in your EasyBite code.

▶ Example of Maths Functions

```
// Example of using various built-in functions in EasyBite

import "Math"

declare x
set x to -5
show abs(x)  // Output: 5

show pow(2, 3)  // Output: 8

show sqrt(16)  // Output: 4

show sin(0)  // Output: 0

show cos(0)  // Output: 1

show tan(0)  // Output: 0

show round(2.6)  // Output: 3

show random()  // Output: A random number between 0 (inclusive) and 1 (exclusive)

show random(1, 10)  // Output: A random integer between 1 (inclusive) and 10
(exclusive)

show max(5, 2, 8, 4)  // Output: 8

show min(5, 2, 8, 4)  // Output: 2

declare arr[5]
set arr to [1, 2, 3, 4, 5]
show sum(arr)  // Output: 15

show ceiling(3.2)  // Output: 4

show floor(3.8)  // Output: 3

show log10(100)  // Output: 2

show average(arr)  // Output: 3

show log(2.718)  // Output: 1

show exp(1)  // Output: 2.718
```

```
show mean(arr)  // Output: 3

declare arr2[7]
set arr2 to [1, 2, 3, 3, 4, 4, 5]
show mode(arr2)  // Output: 3

show sign(-5)  // Output: -1

show log2(8)  // Output: 3
```

String Library

The String library provides various functions for manipulating strings.

Function	Description
`count(str)`	Returns the length of the string `str`.
`contains(str, sub)`	Returns `true` if `sub` is found in `str`, otherwise `false`.
`replace(str, old, new)`	Replaces all occurrences of `old` with `new` in `str`.
`substring(str, start, end)`	Returns the substring of `str` starting from index `start` to `end`.
`uppercase(str)`	Converts `str` to uppercase.
`lowercase(str)`	Converts `str` to lowercase.
`capitalize(str)`	Capitalizes the first letter of each word in `str`.
`strreverse(str)`	Reverses the characters in `str`.
`join(arr, sep)`	Joins the elements of the `arr` into a single string separated by `sep`.
`tolist(str, sep)`	Splits `str` into a list of strings using `sep` as the delimiter.
`compare(str1, str2)`	Compares `str1` and `str2` and returns -1 if `str1` is less, 0 if equal, 1 if `str1` is greater.
`trim(str)`	Removes leading and trailing whitespace from `str`.
`startswith(str, prefix)`	Returns `true` if `str` starts with `prefix`, otherwise `false`.
`endswith(str, suffix)`	Returns `true` if `str` ends with `suffix`, otherwise `false`.
`strremove(str, sub)`	Removes all occurrences of `sub` from `str`.
`split(str, sep)`	Splits `str` into a list of strings using `sep` as the delimiter.
`find(str, sub)`	Returns the index of the first occurrence of `sub` in `str`, or -1 if not found.

These functions allow you to manipulate and extract information from strings in your EasyBite code.

▶ Example of string functions

```
// Example of using additional built-in functions in EasyBite

import "String"

declare str, sub

// count(str)
set str to "Hello, World!"
show count(str)  // Output: 13

// contains(str, sub)
set sub to "World"
show contains(str, sub)  // Output: true

// replace(str, old, new)
set old to "World"
set new to "EasyBite"
show replace(str, old, new)  // Output: "Hello, EasyBite!"

// substring(str, start, end)
show substring(str, 7, 12)  // Output: "World"

// uppercase(str)
set str to "hello, world!"
show uppercase(str)  // Output: "HELLO, WORLD!"

// lowercase(str)
set str to "HELLO, WORLD!"
show lowercase(str)  // Output: "hello, world!"

// capitalize(str)
set str to "hello, world!"
show capitalize(str)  // Output: "Hello, World!"

// strreverse(str)
set str to "Hello, World!"
show reverse(str)  // Output: "!dlroW ,olleH"

// join(arr, sep)
declare arr
set arr to ["Hello", "World", "EasyBite"]
show join(arr, ", ")  // Output: "Hello, World, EasyBite"

// tolist(str, sep)
set str to "Hello, World, EasyBite"
set sep to ", "
show tolist(str, sep)  // Output: ["Hello", "World", "EasyBite"]

// compare(str1, str2)
declare str1, str2
set str1 to "Hello"
```

```
set str2 to "World"
show compare(str1, str2)  // Output: -1

// trim(str)
set str to "   Hello, World!   "
show trim(str)  // Output: "Hello, World!"

// startswith(str, prefix)
set prefix to "Hello"
show startswith(str, prefix)  // Output: true

// endswith(str, suffix)
set suffix to "World!"
show endswith(str, suffix)  // Output: true

// remove(str, sub)
set sub to ", "
show strremove(str, sub)  // Output: "HelloWorld!"

// split(str, sep)
set sep to ", "
show split(str, sep)  // Output: ["Hello", "World", "EasyBite"]

// find(str, sub)
set sub to "World"
show find(str, sub)  // Output: 7
```

Array Library

Provides functions to work with arrays, such as accessing elements, modifying array contents, and performing array operations.

Function	Description
`arr.length()`	Returns the length of the array `arr`.
`arr.append(item)`	Appends `item` to the end of the array `arr`.
`arr.copy()`	Returns a copy of the array `arr`.
`arr.clear()`	Removes all elements from the array `arr`.
`arr.remove(item)`	Removes the first occurrence of `item` from the array `arr`.
`arr.reverse()`	Reverses the order of elements in the array `arr`.
`arr.insert(index, item)`	Inserts `item` at the specified `index` in the array `arr`.
`arr.sort()`	Sorts the elements in the array `arr` in ascending order.
`arr.indexof(item)`	Returns the index of the first occurrence of `item` in the array `arr`, or -1 if not found.

These functions allow you to manipulate and retrieve information from arrays in your EasyBite code.

▶ Example of using additional array-related built-in functions in EasyBite

```
import "Array"
// Example of using additional array-related built-in functions in EasyBite

declare arr

// arr.length()
set arr to [1, 2, 3, 4, 5]
show arr.length()  // Output: 5

// arr.append(item)
arr.append(6)
show arr  // Output: [1, 2, 3, 4, 5, 6]

// arr.copy()
declare arrCopy
set arrCopy to arr.copy()
show arrCopy  // Output: [1, 2, 3, 4, 5, 6]

// arr.clear()
arr.clear()
show arr  // Output: []

// arr.remove(item)
set arr to [1, 2, 3, 4, 5, 6]
arr.remove(3)
show arr  // Output: [1, 2, 4, 5, 6]

// arr.reverse()
arr.reverse()
show arr  // Output: [6, 5, 4, 2, 1]

// arr.insert(index, item)
arr.insert(3, 3)
show arr  // Output: [6, 5, 4, 3, 2, 1]

// arr.sort()
arr.sort()
show arr  // Output: [1, 2, 3, 4, 5, 6]

// arr.indexof(item)
show arr.indexof(4)  // Output: 3
show arr.indexof(7)  // Output: -1
```

Dictionary Library

EasyBite Dictionary Functions

The following functions are available for dictionary operations in EasyBite:

Function	Description
`dictadd(dictionary: dict, key: any, value: any) -> dict`	Adds a key-value pair to the dictionary.
`dictget(dictionary: dict, key: any) -> any`	Retrieves the value associated with a given key from the dictionary.
`dictremove(dictionary: dict, key: any) -> dict`	Removes the key-value pair with the specified key from the dictionary.
`dictcontainskey(dictionary: dict, key: any) -> bool`	Checks if the dictionary contains a specific key.
`dictcontainsvalue(dictionary: dict, value: any) -> bool`	Checks if the dictionary contains a specific value.
`dictsize(dictionary: dict) -> int`	Returns the number of key-value pairs in the dictionary.
`dictkeys(dictionary: dict) -> list`	Returns a list of all keys in the dictionary.
`dictvalues(dictionary: dict) -> list`	Returns a list of all values in the dictionary.
`dictisempty(dictionary: dict) -> bool`	Checks if the dictionary is empty.
`dictclear(dictionary: dict) -> dict`	Removes all key-value pairs from the dictionary.
`dictupdate(dictionary: dict, key: any, value: any) -> dict`	Updates the value associated with a given key in the dictionary.
`dictmerge(dictionary: dict, otherDictionary: dict) -> dict`	Merges another dictionary into the current dictionary.
`dictcopy(dictionary: dict) -> dict`	Creates a shallow copy of the dictionary.
`dicttojson(dictionary: dict) -> str`	Converts the dictionary to a JSON string.
`dicttofile(dictionary: dict, filename: str)`	Writes the dictionary to a file in JSON format.

The EasyBite dictionary functions accept specific parameter types to perform various operations on dictionaries:

- `dictionary` (dict): The dictionary object on which the operation is performed.
- `key` (any): The key used for adding, retrieving, updating, or removing a key-value pair.
- `value` (any): The value associated with the key in the dictionary.
- `otherDictionary` (dict): Another dictionary object to be merged into the current dictionary.
- `filename` (str): The name of the file to which the dictionary is written in JSON format.

The return types of these functions are as follows:

- `dict`: The dictionary object after performing the operation.

- `any`: The value associated with the specified key.
- `bool`: A boolean value indicating the result of the operation.
- `int`: The number of key-value pairs in the dictionary.
- `list`: A list containing either the keys or values from the dictionary.
- `str`: A JSON string representation of the dictionary.

By providing the appropriate parameter types, you can effectively work with dictionaries in EasyBite and perform various operations on them.

▶ Example of Dictionary functions usage

The following functional examples demonstrate the usage of EasyBite dictionary functions:

`dictadd(dictionary: dict, key: any, value: any) -> dict`

Adds a key-value pair to the dictionary.

Example:

```
import "Dict"

set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
show myDict
```

`dictget(dictionary: dict, key: any) -> any`

Retrieves the value associated with a given key from the dictionary.

Example:

```
import "Dict"

set myDict to {} // Empty dictionary
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
set nameValue to dictget(myDict, "name")
show nameValue // Output: Goni
```

`dictremove(dictionary: dict, key: any) -> dict`

Removes the key-value pair with the specified key from the dictionary.

Example:

```
import "Dict"
```

```
set myDict
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
dictremove(myDict, "age")
show myDict
```

`dictcontainskey(dictionary: dict, key: any) -> bool`

Checks if the dictionary contains a specific key.

Example:

```
import "Dict"

set myDict to {}
dictadd(myDict, "name", "Goni")
set containsValue to dictcontainsvalue(myDict, "name")
show containsValue // Output: true
```

`dictcontainsvalue(dictionary: dict, value: any) -> bool`

Checks if the dictionary contains a specific value.

Example:

```
import "Dict"

set myDict to {}
dictadd(myDict, "name", "Goni")
set containsValue to dictcontainsvalue(myDict, "Goni")
show containsValue // Output: true
```

`dictsize(dictionary: dict) -> int`

Returns the number of key-value pairs in the dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
set sizeValue to dictsize(myDict)
show sizeValue // Output: 2
```

`dictkeys(dictionary: dict) -> list`

Returns a list of all keys in the dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
set keysList to dictkeys(myDict)
show keysList // Output: ["name", "age"]
```

`dictvalues(dictionary: dict) -> list`

Returns a list of all values in the dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
set valuesList to dictvalues(myDict)
show valuesList // Output: ["Goni", 25]
```

`dictisempty(dictionary: dict) -> bool`

Checks if the dictionary is empty.

Example:

```
set myDict to {}
set isEmptyValue to dictisempty(myDict)
show isEmptyValue // Output: true
```

`dictclear(dictionary: dict) -> dict`

Removes all key-value pairs from the dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
dictclear(myDict)
show myDict
```

`dictmerge(dictionary: dict, otherDictionary: dict) -> dict`

Merges another dictionary into the current dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")

set additionalDict to {}
dictadd(additionalDict, "city", "New York")
dictadd(additionalDict, "country", "USA")

dictmerge(myDict, additionalDict)
```

`dictcopy(dictionary: dict) -> dict`

Creates a shallow copy of the dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
set copiedDict to dictcopy(myDict)
show(copiedDict)
```

`dicttojson(dictionary: dict) -> str`

Converts the dictionary to a JSON string.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
set jsonString to dicttojson(myDict)
show jsonString // Output: {"name": "Goni", "age": 25}
```

`dicttofile(dictionary: dict, filename: str)`

Writes the dictionary to a file in JSON format.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictadd(myDict, "age", 25)
dicttofile(myDict, "data.json")
```

`dictupdate(dictionary: dict, key: any, value: any) -> dict`

Updates the value associated with a given key in the dictionary.

Example:

```
set myDict to {}
dictadd(myDict, "name", "Goni")
dictupdate(myDict, "name", "Jane")
show myDict
```

DateTime Library

Includes functions to handle date and time operations, such as getting current date and time, formatting dates, and performing date calculations.

Function	Description
`today()`	Returns the current date.
`timenow()`	Returns the current time.
`datediff(date1, date2)`	Calculates the difference in days between `date1` and `date2`.
`dateadd(date, days)`	Adds `days` to the specified `date`.
`dateformat(date, format)`	Formats the `date` according to the specified `format`.
`dateparse(dateString, format)`	Parses the `dateString` using the specified `format` and returns the date.
`timediff(time1, time2)`	Calculates the difference in seconds between `time1` and `time2`.
`timeadd(time, unit, interval)`	Adds the specified `interval` of a given `unit` to the specified `time`.
`timeformat(time, format)`	Formats the `time` according to the specified `format`.
`timeparse(timeString, format)`	Parses the `timeString` using the specified `format` and returns the time.

These functions allow you to perform various operations related to dates and times in your EasyBite code..

▶ Example of using additional date and time-related built-in functions in EasyBite

```
import "DateTime"
// Example of using additional date and time-related built-in functions in
EasyBite

// today()
show today()  // Output: current date
```

```
// timenow()
show timenow()  // Output: current time

// datediff(date1, date2)
declare date1, date2
set date1 to "2022-01-01"
set date2 to today()
show datediff(date1, date2)  // Output: difference in days between date1 and date2

// dateadd(date, days)
set date to today()
set unit to 50
set days to "years"
show dateadd(date, unit, days)  // Output: date after adding days to the specified
date

// dateformat(date, format)
set date to today()
set format to "dd/MM/yyyy"
show dateformat(date, format)  // Output: date formatted according to the
specified format

// dateparse(dateString, format)
set dateString to "01/01/2022"
set format to "dd/MM/yyyy"
show dateparse(dateString, format)  // Output: parsed date from the dateString
using the specified format

// timediff(time1, time2)
declare time1, time2
set time1 to "10:00:00"
set time2 to timenow()
show timediff(time1, time2)  // Output: difference in seconds between time1 and
time2

// timeadd(time, unit, interval)
set time to timenow()
set unit to 30
set interval to "minute"
show timeadd(time, unit, interval)  // Output: time after adding the specified
interval of the given unit

// timeformat(time, format)
set time to timenow()
set format to "HH:mm:ss"
show timeformat(time, format)  // Output: time formatted according to the
specified format

// timeparse(timeString, format)
set timeString to "10:30:00"
set format to "HH:mm:ss"
show timeparse(timeString, format)  // Output: parsed time from the timeString
using the specified format
```

GUI Library

Offers functions for creating graphical user interfaces (GUI), interacting with user inputs, and displaying user interface elements.

Function Name	Description
`createform(formName, width, height)`	Creates a new form with the specified name, width, and height.
`button(formName, buttonText, clickHandler)`	Creates a button on the specified form with the given text and click handler.
`checkbox(formName, controlName, text, isChecked, x, y)`	Creates a check box control on the specified form with the given properties.
`combobox(formName, labelText, top, left, width, height)`	Creates a combo box control on the specified form with the given properties.
`getchecked(formName, controlName)`	Gets the checked state of a check box or radio button control on a form.
`getdock(formName, controlName)`	Gets the docking style of a control on a form.
`getenable(formName, controlName)`	Gets the enabled state of a control on a form.
`getimage(formName, pictureBoxName)`	Retrieves the image from the specified picture box control on the specified form and returns the temporary file path where the image is saved.
`getitem(formName, comboBoxName, index)`	Retrieves the item at the specified index from a combo box control on a form.
`gettext(formName, controlName)`	Retrieves the text of the specified control on the specified form.
`getvalue(formName, progressBarName)`	Gets the current value of a progress bar control on a form.
`getvisible(formName, controlName)`	Gets the visibility of a control on a form.
`getx(formName, controlName)`	Gets the X position of a control on a form.
`gety(formName, controlName)`	Gets the Y position of a control on a form.
`getbackcolor(formName, controlName)`	Gets the background color of a control on a form.
`getforecolor(formName, controlName)`	Gets the foreground color of a control on a form.
`groupbox(formName, text, left, top)`	Creates a group box control on the specified form with the given properties.
`hideform(formName)`	Hides the form with the specified name.

Function Name	Description
`label(formName, text, fontName, fontSize, fontStyle, foreColor, backColor, top, left)`	Creates a label control on the specified form with the given properties.
`listbox(formName, labelText, top, left, width, height)`	Creates a list box control on the specified form with the given properties.
`messagebox(...args)`	Displays a message box with the specified arguments.
`panel(formName, left, top)`	Creates a panel control on the specified form with the given properties.
`progressbar(formName, names, minimum, maximum, value, width, height, top, left)`	Creates a progress bar control on the specified form with the given properties.
`radiobutton(formName, controlName, text, isChecked, x, y)`	Creates a radio button control on the specified form with the given properties.
`runapp(appName)`	Runs the specified application.
`setabove(formName, targetControlName, controlName, spacing)`	Sets the position of the control above another control on the specified form with the given spacing.
`setalignment(formName, controlName, alignment)`	Sets the text alignment of a control on a form.
`setbackcolor(formName, controlName, color)`	Sets the background color of the specified control on the specified form.
`setbelow(formName, targetControlName, controlName, spacing)`	Sets the position of the control below another control on the specified form with the given spacing.
`setchecked(formName, controlName, isChecked)`	Sets the checked state of a check box or radio button control on a form.
`setdock(formName, controlName, dockStyle)`	Sets the docking style of a control on a form.
`setenable(formName, controlName, enable)`	Sets the enabled state of a control on a form.
`setforecolor(formName, controlName, color)`	Sets the foreground color of the specified control on the specified form.
`setimage(formName, pictureBoxName, imagePath)`	Sets the image of the specified picture box control on the specified form using the provided image file path.
`setitem(formName, comboBoxName, item)`	Adds an item to a combo box control on a form.

Function Name	Description
`setleft(formName, targetControlName, controlName, spacing)`	Sets the left position of a control relative to another control on a form.
`setlocation(formName, controlName, x, y)`	Sets the location (X and Y coordinates) of a control on a form.
`setminmax(formName, progressBarName, minimum, maximum)`	Sets the minimum and maximum values of a progress bar control on a form.
`setright(formName, targetControlName, controlName, spacing)`	Sets the right position of a control relative to another control on a form.
`setstyle(formName, controlName, fontName, fontSize, fontStyle, foreColor, backColor)`	Sets the style properties (font, size, style, forecolor, backcolor) of the specified control on the specified form.
`settext(formName, controlName, text)`	Sets the text of the specified control on the specified form.
`setvalue(formName, progressBarName, value)`	Sets the current value of a progress bar control on a form.
`setvisible(formName, controlName, visible)`	Sets the visibility of a control on a form.
`showdialog(dialogName)`	Shows the dialog with the specified name.
`showform(formName)`	Shows the form with the specified name.
`textbox(formName, fontName, fontSize, fontStyle, foreColor, backColor, top, left)`	Creates a text box control on the specified form with the given properties.

Detailed Explanation

button(formName: string, buttonText: string, clickHandler: function)

Creates a button on the specified form with the given text and click handler.

- `formName` (string): The name of the form on which to create the button.
- `buttonText` (string): The text to be displayed on the button.
- `clickHandler` (function): The function that will be called when the button is clicked.

Returns

The created button control.

Example Usage

```
// Create a button on a form named "myForm" with the label "Click Me" and click
handler "HandleButtonClick"

// Example click handler function
function onclickButton()
    // Handle button click logic here
    // ...
end function

button("myForm", "Click Me", onclickButton);
```

checkbox(formName: string, [controlName: string], [text: string], [isChecked: bool], [x: int], [y: int])

Creates a check box control on the specified form with the given properties.

- `formName` (string, required): The name of the form on which to create the check box.
- `controlName` (string, optional): The name of the check box control.
- `text` (string, optional): The text to be displayed next to the check box.
- `isChecked` (bool, optional): The initial checked state of the check box.
- `x` (int, optional): The X position of the check box.
- `y` (int, optional): The Y position of the check box.

Returns

The created check box control.

Example Usage

```
// Create a check box on a form named "myForm" (only required parameter)

checkbox("myForm")

// Create a check box on a form named "anotherForm" with control name "myCheckBox"
//The check box is initially unchecked and has the label "Check me"

checkbox("anotherForm", "myCheckBox", "Check me", false)

// Create a check box on a form named "anotherForm" with control name
"anotherCheckBox"
// The check box is initially checked, has the label "Another check box",
// and is positioned at coordinates (100, 200)

checkbox("anotherForm", "anotherCheckBox", "Another check box", true, 100, 200)
```

combobox(formName: string, [labelText: string], [top: int], [left: int], [width: int], [height: int])

Creates a combobox control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the combobox.
- `labelText` (string, optional): The label text to be displayed above the combobox.
- `top` (int, optional): The top position of the combobox.
- `left` (int, optional): The left position of the combobox.
- `width` (int, optional): The width of the combobox.
- `height` (int, optional): The height of the combobox.

Returns: The created combobox control.

Example Usage:

```
// Create a combobox on a form named "myForm"
combobox("myForm")

// Create a combobox with a label and position on a form named "myForm"
combobox("myForm", "Select an option:", 100, 50)

// Create a combobox with custom dimensions on a form named "myForm"
combobox("myForm", null, null, null, 200, 150)
```

createform(formName: string, width: int, height: int)

Creates a form with the specified name, width, and height.

- `formName` (string): The name of the form to create.
- `width` (int): The width of the form.
- `height` (int): The height of the form.

Returns

The created form.

Example Usage

```
// Create a form named "myForm" with width 800 and height 600

set form to createform("myForm", 800, 600)
runapp(form)
```

getbackcolor(formName: string, controlName: string)

Gets the background color of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The background color of the control as a string value. This can be a color name or a hexadecimal color code.

Example Usage:

```
// Get the background color of a control named "myControl" on a form named
"myForm"
Dim backColor As String = getbackcolor("myForm", "myControl")
Console.WriteLine("Background Color: " & backColor)
```

getdock(formName: string, controlName: string): string

Gets the docking style of a control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control for which to retrieve the docking style.

Returns:

- The docking style of the control as a string. It will be one of the following values:
  - `"None"`: The control is not docked.
  - `"Top"`: The control is docked to the top of the form.
  - `"Bottom"`: The control is docked to the bottom of the form.
  - `"Left"`: The control is docked to the left of the form.
  - `"Right"`: The control is docked to the right of the form.
  - `"Fill"`: The control is docked to fill the remaining space in the form.

Example Usage:

```
// Get the docking style of a control named "myControl" on a form named "myForm"
getdock("myForm", "myControl")
```

getchecked(formName: string, controlName: string)

Gets the checked state of the specified control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control to get the checked state.

Returns:

- (bool) The checked state of the control. `true` if the control is checked, `false` if it is unchecked.

Example Usage:

```
// Get the checked state of a checkbox named "myCheckbox" on a form named "myForm"
set isChecked to getchecked("myForm", "myCheckbox")
```

`getbackcolor(formName: string, controlName: string)`

Gets the background color of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The background color of the control as a string value. This can be a color name or a hexadecimal color code.

Example Usage:

```
// Get the background color of a control named "myControl" on a form named
"myForm"
set backColor to getbackcolor("myForm", "myControl")
show("Background Color: " & backColor)
```

getenable(formName: string, controlName: string): bool

Retrieves the enable state of the specified control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control to retrieve the enable state.

Returns:

- bool: The enable state of the control. `true` if the control is enabled, `false` if it is disabled.

Example Usage:

```
// Get the enable state of a control named "myControl" on a form named "myForm"
set isControlEnabled to getenable("myForm", "myControl")
```

`getforecolor(formName: string, controlName: string)`

Gets the foreground color of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The foreground color of the control as a string value. This can be a color name or a hexadecimal color code.

Example Usage:

```

```

```
// Get the foreground color of a control named "myControl" on a form named
"myForm"
set foreColor to getforecolor("myForm", "myControl")
show("Foreground Color: " + foreColor)
```

`getimage(formName: string, pictureBoxName: string)`

Retrieves the image from a PictureBox control on the specified form.

- `formName` (string): The name of the form containing the PictureBox control.
- `pictureBoxName` (string): The name of the PictureBox control.

Returns: The image object from the PictureBox control, or `null` if no image is set.

Example Usage:

```
// Get the image from a PictureBox named "pictureBox1" on a form named "myForm"
getimage("myForm", "pictureBox1")
```

`getitem(listName: string, index: number): any`

Retrieves the value of an item at the specified index from a list.

- `listName` (string): The name of the list.
- `index` (number): The index of the item to retrieve.

Returns: The value of the item at the specified index.

Example Usage:

```
// Get the value of an item at index 2 from a list named "myList"
getitem("myList", 2)
```

gettext(formName: string, controlName: string)

Retrieves the text of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The text of the control.

Example Usage:

```
// Get the text of a control named "myControl" on a form named "myForm"
gettext("myForm", "myControl")
```

```
```

`getmin(formName: string, controlName: string): int`

Retrieves the minimum value of a control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The minimum value of the control.

Example Usage:

```
// Get the minimum value of a control named "myControl" on a form named "myForm"
getmin("myForm", "myControl")
```

`getmax(formName: string, controlName: string): number`

Retrieves the maximum value of a control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The maximum value of the control.

Example Usage:

```
// Get the maximum value of a control named "myControl" on a form named "myForm"
getmax("myForm", "myControl")
```

`getstyle(formName, controlName)`

Retrieves the style properties (font, size, style, forecolor, backcolor) of the specified control on the specified form.

Parameters:

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control from which to retrieve the style.

Returns:

- `style` (dictionary): A dictionary containing the style properties of the control.

Usage Example:

```
setstyle("MainForm", "label1", "Arial", 12, "Bold", "#FF0000", "#FFFFFF")
setstyle("MainForm", "button1", "Verdana", 10, "Regular", "#000000", "#FFFF00")
```

```
// Get the style of label1
set label1Style to getstyle("MainForm", "label1")
messagebox(label1Style)
// Output: {"font": "Arial", "size": 12, "style": "Bold", "forecolor": "#FF0000",
"backcolor": "#FFFFFF"}

// Get the style of button1
set button1Style to getstyle("MainForm", "button1")
messagebox(button1Style)
// Output: {"font": "Verdana", "size": 10, "style": "Regular", "forecolor":
"#000000", "backcolor": "#FFFF00"}
```

`getvalue(formName: string, controlName: string): any`

Retrieves the current value of a control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.

Returns: The current value of the control.

Example Usage:

```
// Get the value of a control named "myControl" on a form named "myForm"
getvalue("myForm", "myControl")
```

getvisible(formName: string, controlName: string) : bool

Retrieves the visibility state of the specified control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control to retrieve the visibility state.

Returns:

- bool: The visibility state of the control. `true` if the control is visible, `false` if it is not.

Example Usage:

```
// Get the visibility state of a control named "myControl" on a form named
"myForm"
set val to getvisible("myForm", "myControl")
```

getx(formName: string, controlName: string): int

Gets the x-coordinate (horizontal position) of the control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control for which to retrieve the x-coordinate.

Returns:

- The x-coordinate of the control as an integer.

Example Usage:

```
// Get the x-coordinate of a control named "myControl" on a form named "myForm"
getx("myForm", "myControl")
```

gety(formName: string, controlName: string): int

Gets the y-coordinate (vertical position) of the control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control for which to retrieve the y-coordinate.

Returns:

- The y-coordinate of the control as an integer.

Example Usage:

```
// Get the y-coordinate of a control named "myControl" on a form named "myForm"
gety("myForm", "myControl")
```

groupbox(formName: string, [text: string], [left: int], [top: int])

Creates a groupbox control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the groupbox.
- `text` (string, optional): The text to be displayed as the title of the groupbox.
- `left` (int, optional): The left position of the groupbox.
- `top` (int, optional): The top position of the groupbox.

Returns: The created groupbox control.

Example Usage:

```
// Create a groupbox on a form named "myForm"
groupbox("myForm")

// Create a groupbox with a title and position on a form named "myForm"
groupbox("myForm", "Options", 100, 50)
```

hideform(formName: string)

Hides the specified form.

- `formName` (string): The name of the form to be hidden.

Returns: None.

Example Usage:

```
// Hide a form named "myForm"
hideform("myForm")
```

label(formName: string, text: string, [fontName: string], [fontSize: int], [fontStyle: string], [foreColor: string], [backColor: string], [top: int], [left: int])

Creates a label control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the label.
- `text` (string): The text to be displayed on the label.
- `fontName` (string, optional): The font name to be applied to the label.
- `fontSize` (int, optional): The font size (in pixels) to be applied to the label.
- `fontStyle` (string, optional): The font style to be applied to the label. Possible values are "Regular", "Bold", "Italic", or "Underline".
- `foreColor` (string, optional): The foreground color to be applied to the label.
- `backColor` (string, optional): The background color to be applied to the label.
- `top` (int, optional): The top position of the label.
- `left` (int, optional): The left position of the label.

Returns: The created label control.

Example Usage:

```
// Create a label with text "Hello World" on a form named "myForm"
label("myForm", "Hello World")

// Create a label with custom font, size, and color on a form named "myForm"
label("myForm", "Welcome", "Arial", 16, "Bold", "blue")

// Create a label with position and background color on a form named "myForm"
label("myForm", "Label", null, null, null, null, "yellow", 100, 50)
```

listbox(formName: string, [labelText: string], [top: int], [left: int], [width: int], [height: int])

Creates a listbox control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the listbox.

- `labelText` (string, optional): The label text to be displayed above the listbox.
- `top` (int, optional): The top position of the listbox.
- `left` (int, optional): The left position of the listbox.
- `width` (int, optional): The width of the listbox.
- `height` (int, optional): The height of the listbox.

Returns: The created listbox control.

Example Usage:

```
// Create a listbox on a form named "myForm"
listbox("myForm")

// Create a listbox with a label and position on a form named "myForm"
listbox("myForm", "Select an option:", 100, 50)

// Create a listbox with custom dimensions on a form named "myForm"
listbox("myForm", null, null, null, 200, 150)
```

messagebox(formName: string, title: string, message: string, [buttons: string], [icon: string], [defaultButton: string])

Displays a message box with the specified title and message on the specified form.

- `formName` (string): The name of the form on which to display the message box.
- `title` (string): The title of the message box.
- `message` (string): The message to be displayed in the message box.
- `buttons` (string, optional): The label of the button to be shown in the message box. Defaults to an OK button if not provided.
- `icon` (string, optional): The icon to be displayed in the message box. Possible values are "info", "warning", "error", or "question".
- `defaultButton` (string, optional): The label of the button that should be the default button. If not provided, the default button will be determined based on the platform's behavior.

Returns: None.

Example Usage:

```
// Show a message box with a title and message on a form named "myForm"
messagebox("myForm", "Information", "This is an information message.")

// Show a message box with a title, message, and a custom button on a form named
"myForm"
messagebox("myForm", "Confirmation", "Are you sure?", "YesNo")

// Show a message box with a title, message, a custom button, and an icon on a
form named "myForm"
messagebox("myForm", "Error", "An error occurred.", "OKCancel", "error")
```

```
// Show a message box with a title, message, a custom button, an icon, and a
default button on a form named "myForm"
messagebox("myForm", "Question", "Do you want to proceed?", "YesNo", "question",
"No")
```

panel(formName: string, [left: int], [top: int])

Creates a panel control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the panel.
- `left` (int, optional): The left position of the panel.
- `top` (int, optional): The top position of the panel.

Returns: The created panel control.

Example Usage:

```
// Create a panel on a form named "myForm"
panel("myForm")

// Create a panel with a position on a form named "myForm"
panel("myForm", 100, 50)
```

picturebox(formName: string, [names: string], [imagePath: string], [width: int], [height: int], [top: int], [left: int])

Creates a picture box control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the picture box.
- `names` (string, optional): The name of the picture box control.
- `imagePath` (string, optional): The path to the image file to be displayed in the picture box.
- `width` (int, optional): The width of the picture box. Defaults to 100 if not provided.
- `height` (int, optional): The height of the picture box. Defaults to 100 if not provided.
- `top` (int, optional): The top position of the picture box. Defaults to 0 if not provided.
- `left` (int, optional): The left position of the picture box. Defaults to 0 if not provided.

Returns: The created picture box control.

Example Usage:

```
// Create a picture box on a form named "myForm"
picturebox("myForm")

// Create a picture box with custom properties on a form named "myForm"
picturebox("myForm", "picture", "image.png", 200, 150, 50, 50)
```

progressbar(formName: string, [names: string], [minimum: int], [maximum: int], [value: int], [width: int], [height: int], [top: int], [left: int])

Creates a progress bar control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the progress bar.
- `names` (string, optional): The name of the progress bar control.
- `minimum` (int, optional): The minimum value of the progress bar. Defaults to 0 if not provided.
- `maximum` (int, optional): The maximum value of the progress bar. Defaults to 100 if not provided.
- `value` (int, optional): The current value of the progress bar. Defaults to 0 if not provided.
- `width` (int, optional): The width of the progress bar. Defaults to 100 if not provided.
- `height` (int, optional): The height of the progress bar. Defaults to 20 if not provided.
- `top` (int, optional): The top position of the progress bar. Defaults to 0 if not provided.
- `left` (int, optional): The left position of the progress bar. Defaults to 0 if not provided.

Returns: The created progress bar control.

Example Usage:

```
// Create a progress bar on a form named "myForm"
progressbar("myForm")

// Create a progress bar with custom properties on a form named "myForm"
progressbar("myForm", "progress", 0, 100, 50, 200, 30, 50, 100)
```

radiobutton(formName: string, [controlName: string], [text: string], [isChecked: bool], [x: int], [y: int])

Creates a radio button control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the radio button.
- `controlName` (string, optional): The name of the radio button control.
- `text` (string, optional): The text to be displayed next to the radio button.
- `isChecked` (bool, optional): Specifies whether the radio button is checked. Defaults to `false` if not provided.
- `x` (int, optional): The horizontal position of the radio button. If not provided, the control will be placed at its default position.
- `y` (int, optional): The vertical position of the radio button. If not provided, the control will be placed at its default position.

Returns: The created radio button control.

Example Usage:

```
// Create a radio button on a form named "myForm"
radiobutton("myForm")

// Create a radio button with custom properties on a form named "myForm"
radiobutton("myForm", "radioButton", "Option 1", true, 50, 50)
```

runapp(appName: string)

Runs the specified application.

- `appName` (string): The name of the application to run.

Returns: None.

Example Usage:

```
// Run the application named "myApp"

set myApp to createform("myForm", 800, 600)
runapp(myApp)
```

setabove(formName: string, targetControlName: string, controlName: string, [spacing: int])

Sets the specified control above another control on the form with optional spacing.

- `formName` (string): The name of the form on which the controls are located.
- `targetControlName` (string): The name of the control above which the other control should be positioned.
- `controlName` (string): The name of the control to be positioned above the target control.
- `spacing` (int, optional): The spacing between the target control and the control to be positioned above. Defaults to 0 if not provided.

Example Usage:

```
// Set a control named "myControl" above another control named "targetControl" on
a form named "myForm"
setabove("myForm", "targetControl", "myControl")

// Set a control named "myControl" above another control named "targetControl"
with spacing of 10 units on a form named "myForm"
setabove("myForm", "targetControl", "myControl", 10)
```

setalignment(formName: string, controlName: string, alignment: string)

Sets the alignment of a control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `alignment` (string): The desired alignment for the control. Possible values are "left", "center", and "right".

Example Usage:

```
// Set the alignment of a label control named "titleLabel" on a form named
"myForm" to "center"
setalignment("myForm", "titleLabel", "center")
```

`setbackcolor(formName: string, controlName: string, color: string)`

Sets the background color of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `color` (string): The color value to set as the background color. This can be a color name (e.g., "blue") or a hexadecimal color code (e.g., "#0000FF").

Returns: None.

Example Usage:

```
// Set the background color of a control named "myControl" on a form named
"myForm" to blue
setbackcolor("myForm", "myControl", "blue")
```

setbelow(formName: string, targetControlName: string, controlName: string, [spacing: int])

Sets the specified control below another control on the form with optional spacing.

- `formName` (string): The name of the form on which the controls are located.
- `targetControlName` (string): The name of the control below which the other control should be positioned.
- `controlName` (string): The name of the control to be positioned below the target control.
- `spacing` (int, optional): The spacing between the target control and the control to be positioned below. Defaults to 0 if not provided.

Example Usage:

```
// Set a control named "myControl" below another control named "targetControl" on
a form named "myForm"
setbelow("myForm", "targetControl", "myControl")

// Set a control named "myControl" below another control named "targetControl"
with spacing of 10 units on a form named "myForm"
setbelow("myForm", "targetControl", "myControl", 10)
```

setdock(formName: string, controlName: string, dockStyle: string)

Sets the docking style of a control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control for which to set the docking style.
- `dockStyle` (string): The docking style to be set for the control. It should be one of the following values:
  - `"None"`: The control is not docked.
  - `"Top"`: The control is docked to the top of the form.
  - `"Bottom"`: The control is docked to the bottom of the form.
  - `"Left"`: The control is docked to the left of the form.
  - `"Right"`: The control is docked to the right of the form.
  - `"Fill"`: The control is docked to fill the remaining space in the form.

Example Usage:

```
// Set the docking style of a control named "myControl" to "Top" on a form named
"myForm"
setdock("myForm", "myControl", "Top")

// Set the docking style of a control named "myControl" to "Fill" on a form named
"myForm"
setdock("myForm", "myControl", "Fill")
```

setenable(formName: string, controlName: string, enable: bool)

Enables or disables the specified control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control to enable or disable.
- `enable` (bool): The enable state to set for the control. `true` to enable the control, `false` to disable the control.

Returns:

- None.

Example Usage:

```
// Enable a control named "myControl" on a form named "myForm"
setenable("myForm", "myControl", true)
```

`setforecolor(formName: string, controlName: string, color: string)`

Sets the foreground color of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `color` (string): The color value to set as the foreground color. This can be a color name (e.g., "red") or a hexadecimal color code (e.g., "#FF0000").

Returns: None.

Example Usage:

```
// Set the foreground color of a control named "myControl" on a form named
"myForm" to red
setforecolor("myForm", "myControl", "red")
```

`setimage(formName: string, pictureBoxName: string, imagePath: string)`

Sets the image of a PictureBox control on the specified form.

- `formName` (string): The name of the form containing the PictureBox control.
- `pictureBoxName` (string): The name of the PictureBox control.
- `imagePath` (string): The file path of the image to be set.

Example Usage:

```
// Set the image of a PictureBox named "pictureBox1" on a form named "myForm" to
an image file located at "C:\images\image.jpg"
setimage("myForm", "pictureBox1", "C:\images\image.jpg")
```

`setitem(listName: string, index: number, value: any)`

Sets the value of an item at the specified index in a list.

- `listName` (string): The name of the list.
- `index` (number): The index of the item to be set.
- `value` (any): The value to be assigned to the item at the specified index.

Example Usage:

```
// Set the value of an item at index 2 in a list named "myList" to 10
setitem("myList", 2, 10)
```

setleft(formName: string, targetControlName: string, controlName: string, [spacing: int])

Sets the specified control to the left of another control on the form with optional spacing.

- `formName` (string): The name of the form on which the controls are located.
- `targetControlName` (string): The name of the control to the right of which the other control should be positioned.
- `controlName` (string): The name of the control to be positioned to the left of the target control.
- `spacing` (int, optional): The spacing between the target control and the control to be positioned to the left. Defaults to 0 if not provided.

Example Usage:

```
// Set a control named "myControl" to the left of another control named
"targetControl" on a form named "myForm"
setleft("myForm", "targetControl", "myControl")

// Set a control named "myControl" to the left of another control named
"targetControl" with spacing of 10 units on a form named "myForm"
setleft("myForm", "targetControl", "myControl", 10)
```

setright(formName: string, targetControlName: string, controlName: string, [spacing: int])

Sets the specified control to the right of another control on the form with optional spacing.

- `formName` (string): The name of the form on which the controls are located.
- `targetControlName` (string): The name of the control to the left of which the other control should be positioned.
- `controlName` (string): The name of the control to be positioned to the right of the target control.
- `spacing` (int, optional): The spacing between the target control and the control to be positioned to the right. Defaults to 0 if not provided.

Example Usage:

```
// Set a control named "myControl" to the right of another control named
"targetControl" on a form named "myForm"
setright("myForm", "targetControl", "myControl")

// Set a control named "myControl" to the right of another control named
"targetControl" with spacing of 10 units on a form named "myForm"
setright("myForm", "targetControl", "myControl", 10)
```

setstyle(formName: string, controlName: string, [fontFamily: string], [fontSize: int], [fontColor: string], [backgroundColor: string], [borderColor: string])

Applies a style to a control on the specified form.

- `formName` (string): The name of the form.
- `controlName` (string): The name of the control.
- `fontFamily` (string, optional): The font family to be applied to the control.
- `fontSize` (int, optional): The font size (in pixels) to be applied to the control.
- `fontColor` (string, optional): The font color to be applied to the control.
- `backgroundColor` (string, optional): The background color to be applied to the control.
- `borderColor` (string, optional): The border color to be applied to the control.

Returns: None.

Example Usage:

```
// Apply a style to a control named "myControl" on a form named "myForm"
setstyle("myForm", "myControl", "Arial", 14, "black", "white", "gray")
```

settext(formName: string, controlName: string, text: string)

Sets the text of the specified control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `text` (string): The new text to be set on the control.

Returns: None.

Example Usage:

```
// Set the text of a control named "myControl" on a form named "myForm"
settext("myForm", "myControl", "Hello, World!")
```

`setvalue(formName: string, controlName: string, value: any)`

Sets the value of a control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `value` (any): The value to be set for the control.

Example Usage:

```
// Set the value of a control named "myControl" on a form named "myForm" to 10
setvalue("myForm", "myControl", 10)
```

setvisible(formName: string, controlName: string, visible: bool)

Sets the visibility of the control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control for which to set the visibility.
- `visible` (bool): The visibility state to set for the control. `true` to make the control visible, `false` to hide the control.

Returns:

- None.

Example Usage:

```
// Set the visibility of a control named "myControl" on a form named "myForm" to
true
setvisible("myForm", "myControl", true)
```

setx(formName: string, controlName: string, x: int)

Sets the X position of a control on a form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `x` (int): The new X position of the control.

sety(formName: string, controlName: string, y: int)

Sets the Y position of a control on a form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `y` (int): The new Y position of the control.

showdialog(formName: string)

Shows a modal dialog for the specified form.

- `formName` (string): The name of the form for which to show the dialog.

Returns: None.

Example Usage:

```
// Show a modal dialog for a form named "myForm"
showdialog("myForm")
```

setlocation(formName: string, controlName: string, x: int, y: int)

Sets the location (coordinates) of the specified control on the specified form.

- `formName` (string): The name of the form on which the control is located.
- `controlName` (string): The name of the control for which to set the location.
- `x` (int): The X-coordinate of the new location.
- `y` (int): The Y-coordinate of the new location.

Returns:

- None.

Throws:

- `ArgumentException`: If the X or Y values are not valid integers.

Example Usage:

```
// Set the location of a control named "myControl" on a form named "myForm" to
coordinates (100, 200)
setlocation("myForm", "myControl", 100, 200)
```

`setminmax(formName: string, controlName: string, minValue: number, maxValue: number)`

Sets the minimum and maximum values for a control on the specified form.

- `formName` (string): The name of the form containing the control.
- `controlName` (string): The name of the control.
- `minValue` (number): The minimum value to set for the control.
- `maxValue` (number): The maximum value to set for the control.

Example Usage:

```
// Set the minimum and maximum values for a control named "myControl" on a form
named "myForm"
setminmax("myForm", "myControl", 0, 100)
```

`showform(formName: string)`

Displays the specified form.

- `formName` (string): The name of the form to be displayed.

Returns: None.

Example Usage:

```
// Display a form named "myForm"
showform("myForm")
```

textbox(formName: string, [fontName: string], [fontSize: int], [fontStyle: string], [foreColor: string], [backColor: string], [top: int], [left: int])

Creates a textbox control on the specified form with the given properties.

- `formName` (string): The name of the form on which to create the textbox.
- `fontName` (string, optional): The font name to be applied to the textbox.
- `fontSize` (int, optional): The font size (in pixels) to be applied to the textbox.

- `fontStyle` (string, optional): The font style to be applied to the textbox. Possible values are "Regular", "Bold", "Italic", or "Underline".
- `foreColor` (string, optional): The foreground color to be applied to the textbox.
- `backColor` (string, optional): The background color to be applied to the textbox.
- `top` (int, optional): The top position of the textbox.
- `left` (int, optional): The left position of the textbox.

Returns: The created textbox control.

Example Usage:

```
// Create a textbox on a form named "myForm"
textbox("myForm")

// Create a textbox with custom font, size, and color on a form named "myForm"
textbox("myForm", "Arial", 12, "Bold", "black")

// Create a textbox with position and background color on a form named "myForm"
textbox("myForm", null, null, null, null, "lightgray", 100, 50)
```

Files Library

Provides functions to work with files and directories, such as file manipulation, directory operations, and file system access.

Function	Description
`fileappend(filename, content)`	Appends the `content` to the specified `filename`.
`filecopy(source, destination)`	Copies the file from the `source` location to the `destination` location.
`filecreate(filename)`	Creates a new file with the specified `filename`.
`filedelete(filename)`	Deletes the file with the specified `filename`.
`fileexists(filename)`	Checks if the file with the specified `filename` exists.
`filemove(source, destination)`	Moves the file from the `source` location to the `destination` location.
`fileread(filename)`	Reads the content of the specified `filename` and returns it.
`filewrite(filename, content)`	Writes the `content` to the specified `filename`.
`filename(filepath)`	Returns the name of the file from the given `filepath`.
`filepath(filename)`	Returns the path of the file from the given `filename`.
`folderexist(foldername)`	Checks if the folder with the specified `foldername` exists.
`foldername(folderpath)`	Returns the name of the folder from the given `folderpath`.

Function	Description
`folderpath(foldername)`	Returns the path of the folder from the given `foldername`.
`getfileextension(filename)`	Retrieves the file extension from the given `filename`.
`getfiles(foldername)`	Returns a list of files in the specified `foldername`.
`getfolders(foldername)`	Returns a list of folders in the specified `foldername`.
`getlastmodifiedtime(filename)`	Retrieves the last modified timestamp of the file with the given `filename`.
`getparentdirectory(path)`	Retrieves the parent directory path from the given `path`.
`getfilesize(filename)`	Returns the size of the file in bytes for the given `filename`.
`getsub(foldername)`	Returns a list of sub-folders in the specified `foldername`.
`makefolder(foldername)`	Creates a new folder with the specified `foldername`.
`movefolder(source, destination)`	Moves the folder from the `source` location to the `destination` location.
`readcontent(filename)`	Reads content of `file` and return all lines in array
`readline(filename, lineNumber)`	Reads the line at the specified `lineNumber` from the file with the given `filename`.
`readline(filename, start, end)`	Reads the line at the specified `range` by given `start` and `end` from the file with the given `filename`.

These functions provide convenient ways to perform file and folder operations in your EasyBite code.

▶ Examples of the File function

Examples

```
// Example of using additional file and folder-related built-in functions in
EasyBite

import "Files"

// filewrite(filename, content)
set filename to "example.txt"
set content to "Hello, World!"
filewrite(filename, content)

// fileread(filename)
declare content
set content to fileread(filename)
show content

// fileappend(filename, content)
```

```
set content to "This is additional content."
fileappend(filename, content)

// filecreate(filename)
set newFilename to "newfile.txt"
filecreate(newFilename)

// fileexists(filename)
show fileexists(filename)
show fileexists("nonexistent.txt")

// readline(filename, lineNumber)
set lineContent to readline(filename, 1)
show lineContent

// readlines(filename, start, end)
set lineContents to readlines(filename, 2, 3)
show lineContents

// readcontent(filename)
set rcontent to readcontent(filename)
for i from 0 to rcontent.length()
    show(rconent[i])
end for

// filedelete(filename)
filedelete(newFilename)

// getfiles(foldername)
set foldername to "path/to/folder"
declare files
set files to getfiles(foldername)
show files

// filename(filepath)
set filepath to "path/to/folder/example.txt"
show filename(filepath)

// filepath(filename)
set filename to "example.txt"
show filepath(filename)

// filemove(source, destination)
set source to "path/to/folder/example.txt"
set destination to "path/to/newfolder/example.txt"
filemove(source, destination)

// filecopy(source, destination)
set source to "path/to/folder/example.txt"
set destination to "path/to/newfolder/example_copy.txt"
filecopy(source, destination)

// makefolder(foldername)
set foldername to "newfolder"
```

```
makefolder(foldername)

// folderexist(foldername)
show folderexist(foldername)
show folderexist("nonexistent_folder")

// foldername(folderpath)
set folderpath to "path/to/folder/subfolder"
show foldername(folderpath)

// folderpath(foldername)
set foldername to "subfolder"
show folderpath(foldername)

// getsub(foldername)
set foldername to "path/to/folder"
declare subfolders
set subfolders to getsub(foldername)
show subfolders

// getfolders(foldername)
set foldername to "path/to/folder"
declare folders
set folders to getfolders(foldername)
show folders

// getfileextension(filename)
set filename to "example.txt"
declare extension
set extension to getfileextension(filename)
show extension

// getfilesize(filename)
set filename to "example.txt"
declare size
set size to getfilesize(filename)
show size

// getlastmodifiedtime(filename)
set filename to "example.txt"
declare modifiedTime
set modifiedTime to getlastmodifiedtime(filename)
show modifiedTime

// getcreationtime(filename)
set filename to "example.txt"
declare creationTime
set creationTime to getcreationtime(filename)
show creationTime

// getparentdirectory(path)
set path to "path/to/folder/example.txt"
declare parentDirectory
set parentDirectory to getparentdirectory(path)
```

```
show parentDirectory

// movefolder(source, destination)
set source to "path/to/folder"
set destination to "path/to/newfolder"
movefolder(source, destination)
```

In this example, each file and folder-related function is used with specific file names and folder names to demonstrate its functionality. The output of each function is displayed using the `show` statement. Please note that the paths and file/folder names used in the example are placeholders and should be replaced with actual file/folder names in your implementation.

Misc Library

Includes miscellaneous functions that do not fit into other specific libraries, covering various utility functions and operations. You can access it by `import "Misc"`

Function	Description
`toint(value)`	Converts the `value` to an integer.
`todouble(value)`	Converts the `value` to a double.
`tostring(value)`	Converts the `value` to a string.
`isint(value)`	Checks if the `value` is an integer.
`isalnum(value)`	Checks if the `value` is alphanumeric.
`isdigit(value)`	Checks if the `value` is a digit.
`isdouble(value)`	Checks if the `value` is a double.
`isstring(value)`	Checks if the `value` is a string.
`islist(value)`	Checks if the `value` is a list.
`isdict(value)`	Checks if the `value` is a dictionary.
`typeof(value)`	Returns the type of the `value`.

These functions are useful for performing type conversions and type checking operations in your EasyBite code.

▶ Example of using Misc additional type-related built-in functions in EasyBite

---

Examples

```
// toint(value)
set value to "10"
set intValue to toint(value)
```

```
show intValue  // Output: 10

// todouble(value)
set value to "3.14"
set doubleValue to todouble(value)
show doubleValue  // Output: 3.14

// tostring(value)
set value to 42
set stringValue to tostring(value)
show stringValue  // Output: "42"

// isint(value)
set value to 10
show isint(value)  // Output: true

// isalnum(value)
set value to "abc123"
show isalnum(value)  // Output: true

// isdigit(value)
set value to "123"
show isdigit(value)  // Output: true

// isdouble(value)
set value to 3.14
show isdouble(value)  // Output: true

// isstring(value)
set value to "Hello, World!"
show isstring(value)  // Output: true

// islist(value)
set values to [1, 2, 3]
show islist(values)  // Output: true

// isdict(value)
set value to {"key": "value"}
show isdict(value)  // Output: true

// typeof(value)
set value to "Hello, World!"
show typeof(value)  // Output: "string"
```

SQLite Library

SQLite is a lightweight, embedded relational database management system that provides a simple and efficient way to store, retrieve, and manage data within applications. It offers a set of functions and APIs that allow developers to interact with SQLite databases programmatically.

These functions enable you to perform various database operations using SQLite. You can establish a connection to a database, execute SQL queries, fetch and manipulate data, manage transactions, and retrieve database information. The `sqliteconnect()` function is used to establish a connection to a specific SQLite database file by providing the database name (`dbname`).

The `sqlite_query()` function is used to execute SQL queries on the established connection. It takes the connection object (`connection`) and the query string (`query`) as parameters and returns the result as a `SQLiteDataReader` object.

To retrieve data from the result set, you can use functions such as `sqlite_fetchall()`, which fetches all rows as a list of dictionaries, `sqlite_fetchassoc()`, which fetches the next row as a dictionary, or `sqlite_fetchrow()`, which fetches the next row as an array of objects.

The `sqlite_fetcharray()` function fetches the next row from the SQLite result set `result` and returns it as an array.

Other functions like `sqlite_insertid()` retrieve the last inserted row ID, `sqlite_close()` closes the database connection, and `sqlite_commit()` and `sqlite_rollback()` handle transaction management.

Additionally, functions like `sqlite_escape_string()` help escape special characters in input strings for safe use in SQLite queries. You can also retrieve the SQLite version using `sqlite_version()` and create new database files using `sqlite_create(dbname)`.

These SQLite functions provide a powerful and efficient way to work with SQLite databases, allowing you to store and retrieve data seamlessly within your applications.

SQLite Functions

The following functions are available for SQLite database operations:

Function	Description
`sqliteconnect(dbname)`	Establishes a connection to the SQLite database specified by `dbname`.
`sqlite_numrows(result)`	Returns the number of rows in the SQLite result set `result`.
`sqlite_query(connection, query)`	Executes the SQL query specified by `query` on the SQLite `connection` and returns the result as a `SQLiteDataReader` object.
`sqlite_fetchall(result)`	Fetches all rows from the SQLite result set `result` and returns them as a list of dictionaries.
`sqlite_fetchassoc(result)`	Fetches the next row from the SQLite result set `result` and returns it as a dictionary.
`sqlite_fetchrow(result)`	Fetches the next row from the SQLite result set `result` and returns it as an array of objects.
`sqlite_fetcharray(result)`	Fetches the next row from the SQLite result set `result` and returns it as an array.
`sqlite_insertid()`	Returns the last inserted row ID in the SQLite database.

Function	Description
`sqlite_close()`	Closes the SQLite database connection.
`sqlite_commit()`	Commits the current transaction in the SQLite database.
`sqlite_begin_transaction()`	Begins a new transaction in the SQLite database.
`sqlite_rollback()`	Rolls back the current transaction in the SQLite database.
`sqlite_escape_string(input)`	Escapes special characters in the input string `input` for safe use in SQLite queries.
`sqlite_error()`	Returns the last SQLite error message, if any.
`sqlite_version()`	Returns the version of SQLite being used.
`sqlite_create(dbname)`	Creates a new SQLite database file with the specified `dbname`.

▶ Example of SQLite functions usage

```
// Import SQLite library
import "SQLite"


// Establish a connection to the SQLite database
declare connection
sqlite_create("mydatabase.db")
set connect to sqliteconnect("mydatabase.db")

declare createTableQuery
set createTableQuery to "CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY
AUTOINCREMENT, name TEXT, age INTEGER)"
set query to sqlite_query(connect, createTableQuery)

// Insert data into the table
if (query != false) then

    declare insertQuery
    set insertQuery to "INSERT INTO users (name, age) VALUES ('Goni Doe', 25)"

    if (sqlite_query(connect, insertQuery) != false) then

           show("Record Inserted Successfully")
    else
        show("Record Not Inserted")
    end if
else

 show("An error occured: query not executed")

end if
```

```
declare selectQuery
set selectQuery to "SELECT * FROM users"
declare result
set result to sqlite_query(connect, selectQuery)

// Get the number of rows in the result set

// Get the number of rows in the result set
declare numRows
set numRows to sqlite_numrows(result)
show(numRows)


// Fetch the next row from the result set as a dictionary
declare nextRow
set row to sqlite_fetchassoc(result)

// Print the data from the dictionary
    show(row["name"])
    show(row["age"])

set nextRow to sqlite_fetcharray(result)
show(nextRow[0])
show(nextRow[1])
// Print the data from the dictionary

// Get the last inserted row ID
declare lastInsertID
set lastInsertID to sqlite_insertid()
show(lastInsertID)

// Close the database connection
sqlite_close()
```

Contact for Feedback and Bug Reports

- **Email**: You can reach out to me directly via email at muhammadgoni51@gmail.com for any feedback, suggestions, or bug reports related to EasyBite. Please use a descriptive subject line to ensure your message gets attention.

- **Issue Tracker**: If you encounter any bugs or would like to request new features, please use the GitHub issue tracker in this repository. Follow these steps to create a new issue:

    1. Go to the Issues tab of this repository.
    2. Click on the "New Issue" button.
    3. Provide a clear and concise title for the issue.
    4. In the issue description, explain the problem or suggestion in detail. Include any relevant information such as steps to reproduce the issue, error messages, or screenshots if applicable.
    5. Choose the appropriate issue template if available, or leave it as a blank issue.
    6. Click on "Submit new issue" to create the issue.

- **Community Forum**: Join our dedicated EasyBite community forum to connect with other users, ask questions, and engage in discussions. Visit the EasyBite Community Forum to register and participate. We encourage you to share your experiences, ideas, and feedback with the community.

Bug Reporting Guidelines

To ensure efficient bug tracking and resolution, please follow these guidelines when submitting bug reports:

- Provide detailed steps to reproduce the issue. Include the version of EasyBite you are using, your operating system, and any relevant error messages or logs.
- If possible, isolate the issue and create a minimal example that demonstrates the problem.
- Attach any relevant files or screenshots that can help us understand and reproduce the issue.
- If you encounter multiple issues, please create separate bug reports for each.

Thank you for your help in improving EasyBite! Your feedback is highly valuable to us.

Code of Conduct

We value an inclusive and respectful community. Please review our Code of Conduct to understand the guidelines and expectations for participating in the EasyBite community. We strive to maintain a safe and welcoming environment for all contributors.

Continuous Monitoring and Engagement

I will actively monitor the provided communication channels, including email and the issue tracker, to respond to your feedback, bug reports, and observations. I aim to provide timely assistance, updates, and resolutions to ensure the best experience with EasyBite.