Syntax

Statement	Syntax	Reason
begin of scope	{	Uses the syntax of the most common modern programming languages - fast to write (one char)
end of scope	}	

while	The while statement executes an Expression and a Statement repeatedly until the value of the Expression is false
	
	`int a`
	`a := 0`
	
	`while (expression) {`
	`        statement`
	`}`
		Makes sense in form of english gramma notation and used in most languages today

switch	The switch statement transfers control to one of several statements depending on
	
	the value of an expression. When one case been executed the switch is exited.
	
	`switch(expression) {`
	`        case a: statement1`
	`(implicit break)`
	`        case b:  statement2`
	`(implicit break)`
	
	`        default statement3`
	`(implicit break)`
	`}`
		Switch statements are very useful in the Arduino environment - automatically provides a break as you don't want users to accidentally fall-through.

AND	The conditional-and operator AND evaluates true if both values are true. `true AND true`	Common logical statement - must have

OR	The conditional-or operator OR evaluates true if either of the values are true or false.
	
	`true OR false`
	`true OR true`
		Common logical statement - must have

else	If the value is true, then the first contained Statement (the one before the else keyword) is executed.
	
	`if (expression) {`
	`        statement`
	`}`
	
	`else {`
	`        statement`
	`}`
		Common logical statement - must have

if	The if statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.
	
	The Expression must have type boolean or Boolean, or a compile-time error occurs.
	
	`if(expression) {`
	`        statement`
	`}`
		Common logical statement - must have

plus	The unary plus operator + `(1 + 1) = 2`	Common logical statement - must have
minus	The unary minus operator - `(1 - 1) = 0`	Common logical statement - must have
divide	`(10 / 2) = 5`	Common logical statement - must have
multiple	`(5 * 2) = 10`	Common logical statement - must have
int	`intvar1 := 1`	Common logical statement - must have
double	`doublevar1 := 1.1`	No need for float Common logical statement - must have.

list	list ava[2] := (1, 2)
	
	listAdd(list, Value, index)
	
	listRemove(list, Value, index)
		To provide an easier way of using arrays, in form of what modern languages do

boolean	A boolean can either be true or false
	
	`var1 := true`
	`var2 := false`
		Common logical statement - must have

string	`var1 := "abc"`	Common logical statement - must have " rather than ' in order to use english grammar like don't

func
 	
functuin arguments
 	
function type
 	
return value
	`func int function1 (int var1,`
	`int var2) {`
	`        return var1 + var2 + 1;`
	`}`
	
	`func function2 () {`
	`        printf("Hello World")`
	`}`
		Short for function. Implicit void function type unless other data type specified

scope	Nested scoping	Monolitic (Global scoping) not really useful. Flat stucture does not give enough flexibility in terms of declarations of variables. With a flat structure, you cannot have a function within a function.
custom error handling (try catch exceptions)	None	
EOL	No end of line symbol	
assignment	:=	Mathematical notation Differs from the single =, making the users aware.
less than <	The numerical comparison operators <, <=, >, and >= `1 < 2`	logical
larger than	> `2 > 1`	logical

less than or equal	<=
	
	`1 <= 2`
	`1 <= 1`
		logical

larger than or equal	>=
	
	`2 >= 1`
	`1 >= 1`
		logical

equal	Equal operator that returns true when the left hand expression is equal to the right hand expression
	
	=
	
	`1 = 1`
	`true = true`
	`false = false`
		logical

not equal	Not equal operator != that returns true when the left hand expression is not equal to the right hand expression `1 != 2` `false = true` `false = false`	Logical common
not	The not expression !, negates the expressions resulting boolean. `!(false) = true`	logical
comments	Comments are single line and written after #. `#comment example`	Single char.
print	print a text formatted (more specification needed) `print("Hello World")`	