TinyScript specifikáció

Variables

Types

- integer
- boolean
- string

Declaration

type name;
Where type can be integer, boolean or string
or
var name;

Assigning values

Assigning values to variables can be done with the = operator.
name **=** value;
value can be null, an other variable or an expression

Supported operators

Integer

+	Addition
-	Substraction
*	Multiplication
/	Division
<	Lesser
<=	Lesser or equal
==	Equal
!=	Not Equal
>	Greater
>=	Greater or equal

Boolean

+	Or
*	And
!	Not
==	Equal
!=	Not equal

String

+	Addition
==	Equal
!=	Not equal

Arrays

Declaration

type name[x];
Where type can be integer, boolean or string.
x represents the number of elements in the array.

or

type name[] = {value1, value 2, value3};
Creates a 3 sized array with:

- value1 at index 0
- value2 at index 1
- value3 at index 2

Assigning values

name[x] = value;
Where x is the index of the element in the array.
Indexing of the elements starts from 0.

Example

int myArray[10];	Creates an array with 10 elements.
myArray[5] = 0;	Sets the sixth element to 0.

Loops

While

The given expression must be a boolean value.
The while loop runs when the expression is true.

```
while (expression) {
        statement1;
        statement2;
}
```

For

```
for (initialization; condition; afterthought){
  statement;
}
```

Example: Prints the numbers from 1 to 10 with the increment of 1.

```
for (x = 1; x<=10;x++){
  print(i);
}
```

Count

```
count (initialization; afterthought){
  statement;
}
```

Example: Prints the numbers from 1 to 10 with the increment of 1.

```
count(from x = 1 to 10;x++){
  print(i);
}
```

Do While

The given expression must be a boolean value.
The while loop runs while the expression is true.
It runs at least on time no matter if the expression is true or not.

```
do{
  statement;
} while (expression);
```

Conditional statement → if - else

The given expression must return a boolean value.
If the expression is true, then the statement is executed.

```
if (expression) {
        statement;
}

if (expression) {
        statement1;
}
else {
        statement2;
}
```

Outputs

print

```
print(string);
print("BME AUT");
```

Prints a string to the standard output.

Inputs

read

```
read(myVariable);
```

Reads data from the standard input to the given variable.
The type of the variable determines how it interprets the characters.

Functions

Inbuilt functions

abs

Returns the absolute value of the given number.

abs(number);

max

Returns the maximum value of the given numbers.

max(number1,number2,number3);

min

Returns the minimum value of the given numbers.

min(number1,number2,number3);

User defined functions

Function declarations

```
return_type function_name( parameter list ) {
  body of the function
}
```

- Return Type – A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- Function Name – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- Parameters – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- Function Body – The function body contains a collection of statements that define what the function does.

Calling a function

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.