Expressions:

a + b * c - d //returns b*c+a-d

Expressions support operands ‘+’ ‘-’ ‘*’ ‘/’. They do not support ‘(‘ or ‘)’.

Example:
1 + 3 //Good. Returns 4.
(1+2)*2 //Bad. Undefined behaviour. Do not use “(“ or “)”.
1*2+2*2 //Good. Returns 6.
-1 //Good. Returns -1.

Initialization & Assignment:

n= k //creates or assigns value k to variable n

Assignment is an initialization. There is no difference between them.

Example:
n= 5 //n=5
n= n * n //n=25
n = 1 //n=1

Arrays:

n[k] // returns value under n[k]

Arrays are actually dynamically calculated variable names.So n[3+1] gets interpreted as n[4]. For the end user, there is no difference between this and your typical arrays.

Arrays **do not** support nested indexes n[a[k]] **is NOT allowed**.

Examples:
n[1]= 1 //creates variable n[1] and assigns 1 to it
n[2 - 1]= 0 // n[1] = 0
n[n[1]] = 0 //Undefined behaviour.

Compare statements:

(a.b) where '.' is any compare statement.

Supported Compare Statements:
(a==b)
(a!=b)
(a>=b)
(a<=b)
(a.b && c.d || e.f) // and so on. There is no limit on putting more && and ||.

Note that expressions are not implicitly evaluated as true.(1) doesn't return true.

Loops:

while(azb) -- executes code while comp_statement returns true
//code
end while

Example:

n= 3
while(n*2<10)
n = n + 1
end while

//// n is now equal to 5 because 5*2 is not smaller than 10

If statements:

If(azb) --executes code if com_statement retruns true
//code
end if

Swap:

swap(a,b) - swaps values of 2 variables

Example:

a = 1
b = 2
c = 3

ar[1] =11
ar[b] = 12 //Ar[2] = 12

swap(a,b) //b= 1, a=2
swap(Ar[1],Ar[b]) //Ar[1] = 12, Ar[2] = 11

Writing code:

Every code is implicitly started with below instructions:

s= <array size>
ar[0] = a
ar[1] = b
.
.
ar[s-1] = x

where a,b...x are corresponding array values imported to the interpreter.
Every code must end with “end program line”.

Our comparisons between 2 array elements are highlighted with yellow border around arrays.

```
if(a[i]>=1)                 //doesn’t highlight anything
while(a[i]>=a[i+1])         // highlights a[i] and a[i+1]
```

Our swaps between 2 array elements are highlighted with yellow border around number boxes and animated.

```
swap(a[i+1], a[i])          //animates the swap
```

```
v= a[i+1]
a[i+1] = a[i]
a[i] = v                    //does the same thing but doesn’t animate/highlight the swap
```

Sample codes:

1/ Bubble Sort:

```
procedure bubbleSort( A : lista elementów do posortowania )
  n = liczba_elementów(A)
  do
    for (i = 0; i < n-1; i++) do:
      if A[i] > A[i+1] then
        swap(A[i], A[i+1])
      end if
    end for
    n = n-1
  while n > 1
end procedure
```

```
    while(s>1)                          //remember s is array size
            i= 0
            while(i<s - 1)
                    if(ar[i]>ar[i+1])           //remember the array is evaluated with ar[x]
                            swap(ar[i],ar[i+1])
                    end if
            i= i + 1
            end while
            s=s - 1
    end while
end program                             //the program must end with end program line
```

2/ Insertion Sort

```
0. Insert_sort(A, n)

1.  for i=2 to n :
2       # Wstaw A[i] w posortowany ciąg A[1 ... i-1]
3.      wstawiany_element = A[i]
4.      j = i - 1
5.      while j>0 and A[j]>wstawiany_element:
6.          A[j + 1] = A[j]
7.          j = j - 1
8.      A[j + 1] = wstawiany_element
```

```
i=1
while(i<s)

        j= i-1
                        while(j>=0 && ar[j] > ar[j+1])
                                swap(ar[j+1],ar[j])
                                j= j-1
                        end while

                i= i+1
end while
end program
```

Note in the above example we work hard to implement comparisons and swaps between array elements instead of a helper variable.