

ELPL



```
let x be 5
let y be 10
```

```
print "Initial values: " x " and " y
```

```
if x is less than y then{
  print "x is less than y"
}else{
  print "x is not less than y"
}
```

```
repeat 3 times {
  print "Repeating" x
}
```

```
while x is less than 10 {
  print "Incrementing x:" x
  let x be x add 1
}
```

```
function greet() {
  print "Hello from function!"
}
```

```
call greet()
```

```
let result be y multiply x
print "x * y = " result
```

```
let condition be true
if condition is equal to true then{
  print "Condition is true"
}else{
  print "Condition is false"
}
let a be not false
print "a (not false) = " a
```

```
Array nums be [1, 2, 3, 4]
```

```
let sums be 0
```

```
for i be 0 to length(nums) {  
    let sums be sums add nums[i]  
}
```

```
print sums / prints: 10
```

```
print nums[1] / prints: 2
```

```
/ this is a single line comment  
>This is a multi line  
comment<
```

```
---in newer version above 6.0.1 Beta----  
// single comment  
>>> Multi-line<<<
```

```
-----
```

```
for i be 0 to 5 {  
    print "Loop at i ="  
    print i  
    if i is equal to 3 then {  
        print "Stopping at i = 3"  
        stop  
    }  
}  
print "Loop ended"
```

```
---output below---
```

```
Loop at i =  
0  
otherwise case  
Loop at i =  
1  
otherwise case  
Loop at i =  
2  
otherwise case  
Loop at i =  
3  
Stopping at i = 3  
Loop ended
```

```
-----
```

```
Array nums be [2,7,11,15]
```

```
let target be 9
```

```
for i be 0 to 3 {  
    for j be 1 to 3{  
        if nums[i] add nums[j] is equal to target then{
```

```

    print "Indices: "
    print i j
    stop / for 'break'
} otherwise {
    print "no found"
}
}
stop
}

```

RECURSION

```

function factorial(n) {
    if n is equal to 0 then {
        return 1
    }
    return n multiply factorial(n subtract 1)
}

```

call factorial(5) / prints: 120

RECURSION USING ARRAYS

Array memo be [0,0,0,0,0,0,0,0]

```

function fibonacci(n) {
    if n is less than 2 then {
        return n
    }

    if not (memo[n] is equal to 0) then {
        return memo[n]
    }

    let a be call fibonacci(n subtract 1)
    let b be call fibonacci(n subtract 2)
    let memo[n] be a add b
    return memo[n]
}

```

print "Fibonacci(6) = " fibonacci(6) / Fibonacci(6) = 8

Solving twoSum

Array nums be [2,7,11,15]

let target be 9

```

for i be 0 to 3{
  for j be 1 to 3 {
    if nums[i] add nums[j] is equal to target then {
      print "indices: " i j
      stop / will break the loop
    } otherwise {
      print "no pair found"
    }
  }
  stop
} / prints : indices : 0 1

```

```

Array nums be [1,1,1,1]

```

```

let sums be 0
for i be 0 to length(nums){
  let sums be sums add nums[i]
}
print sums / output : 4
-----

```

LOGICALS

```

let flag be false

if not flag then {
  print "Flag is false"
} otherwise {
  print "Flag is true"
}

```

```

let a be true
let b be false

if a and not b then {
  print "Logical expression works!"
}

```

solving twoSum using Built-In function length

```

Array nums be [2, 7, 11, 15]

```

```

let target be 9
let found be false

```

```

for i be 0 to length(nums){

```

```

for j be i add 1 to length(nums){
  let sums be nums[i] add nums[j]
  print "checking pair: " i j " sum: " sums
  if sums is equal to target then {
    print "indices: " i j
    let found be true
    stop
  } otherwise{
    print "pair no found"
  }
}

stop
}

```

```

-----
Array nums be [2, 7, 11, 15]
let target be 9
let found be false
Array result be [-1, -1]

for i be 0 to length(nums) subtract 1 {
  for j be i add 1 to length(nums) subtract 1 {
    let sum be nums[i] add nums[j]
    print "checking pair:" i j "sum:" sum

    if sum is equal to target then {
      let found be true
      let result[0] be i
      let result[1] be j
      stop
    }
  }
}
if found is equal to true then {
  stop
}
}

```

```

if found is equal to true then {
  print "indices: " result[0] result[1]
} otherwise {
  print "no pair found"
}

```

REVERSING AN ARRAYS

```

Array nums be [3, 9, 12,11]
Array reversed be []

```

```

for i be 0 to length(nums){
  let reversed[i] be nums[length(nums) subtract 1 subtract i]
}

```

print reversed

IN place reversal

```
Array nums be [2, 7, 11, 15]
let n be length(nums)
let mid be n divide 2

for i be 0 to mid subtract 1 {
    let temp be nums[i]
    let nums[i] be nums[n subtract 1 subtract i]
    let nums[n subtract 1 subtract i] be temp
}

print nums
```

Recursion sum of Arrays

```
function sumArray(arr, i) {
    if i is equal to length(arr) then {
        return 0
    }
    return arr[i] add call sumArray(arr, i add 1)
}

Array nums be [1, 2, 3, 4, 5]
call sumArray(nums, 0)
```

More recursive function

```
function power(x, n) {
    if n is equal to 0 then {
        return 1
    }
    return x multiply call power(x, n subtract 1)
}

call power(2, 4) / should print 16
```

threeSum Solution

```
Array nums be [-1, 0, 1, 2, -1, -4]

for i be 0 to length(nums) subtract 3 {
    for j be i add 1 to length(nums) subtract 2 {
        for k be j add 1 to length(nums){
            let sums be nums[i] add nums[j] add nums[k]
```



```

    for k be j add 1 to length(nums){
      for l be k add 1 to length(nums) {
        let sums be nums[i] add nums[j] add nums[k] add nums[l]
        print "checking quadruple: " i j k l " sum: " sums
        if sums is equal to target then {
          print "FOUND quadruple at indices: " i j k l " sum: "
sums
          stop
        } otherwise {
          print "not a valid quadruple"
        }
      }
    } stop
  } stop
} stop
}

```

----- **fiveSum solution**

```

Array nums be [-2, -1, 0, 1, 2, 3]
let target be 3

```

```

for i be 0 to length(nums) subtract 5 {
  for j be i add 1 to length(nums) subtract 4 {
    for k be j add 1 to length(nums) subtract 3 {
      for l be k add 1 to length(nums) subtract 2 {
        for m be l add 1 to length(nums){
          let sums be nums[i] add nums[j] add nums[k] add nums[l] add
nums[m]
          print "checking quintuple: " i j k l m " sum: " sums
          if sums is equal to target then {
            print "FOUND quintuple at indices: " i j k l m " sum: " sums

            stop
          } otherwise {
            print "not found"
          }
        }
      }
    } stop
  }
} stop
} stop
}

```

----- **Recursion**


```
function fib(n){  
  if n is less than or equal to 1 then {  
    return n  
  } otherwise {  
    let a be fib(n subtract 1)  
    let b be fib(n subtract 2)  
    return a add b  
  }  
}
```

```
}
```

```
for i be 0 to 30{  
  print call fib(i)  
}
```

0

1

1

2

3

5

8

13

21

34

55

89

144

233

377

610

987

1597

2584

4181

6765

10946
17711
28657
46368
75025
121393
196418
317811
514229
832040

Backtracking and recursion

```
function subsetSum(nums, index, target) {  
    print"subsetSum called with index = " index ", target = " target  
  
    if target is equal to 0 then {  
        print"Target reached 0 – returning true"  
        return true  
    }  
  
    if index is equal to length(nums) then {  
        print"Reached end of array – returning false"  
        return false  
    }  
  
    if nums[index] is less than or equal to target then {  
        print"Trying including nums[" index "] = " nums[index]  
        if call subsetSum(nums, index add 1, target subtract nums[index]) then {
```

```
        print"Path including nums[" index "]" worked – returning true"
        return true
    }
}
```

```
print"Trying excluding nums[" index "]" = " nums[index]
let result be call subsetSum(nums, index add 1, target)
print"Result after excluding nums[" index "]: " result
return result
}
```

```
Array nums be [3, 4, 5, 2]
let target be 9
let result be call subsetSum(nums, 0, target)
print"Final result: " result
```

Recursion fibonacci

```
function fib(n) {
    if n is equal to 0 then {
        return 0
    }
    if n is equal to 1 then {
        return 1
    }
    let a be fib(n subtract 1)
    let b be fib(n subtract 2)
    return a add b
}
```

```
let result be fib(6)
print "Fib(6): " result
```

Palindrome

```
function isPalindrome(s, left, right) {
  if left is greater than or equal to right then {
    return true
  }

  if s[left] not equal to s[right] then {
    return false
  }

  return isPalindrome(s, left add 1, right subtract 1)
}

/ Test case
Array word be ["r", "a", "c", "e", "c", "a", "r"]
let result be isPalindrome(word, 0, length(word) subtract 1)
print "Is Palindrome: " result
```

2D Arrays

```
Array nums be [[0,2,3],[2,3,4]]

let total be 0

for i be 0 to length(nums){
  for j be 0 to length(nums[i]){
    let total be total add nums[i][j]
  }
}

print total / output: 14
```

Maps

```
Map[key,value] mymap be {1: "a", 2: "b", 3: "c", 5: "v"}

foreach k,v : mymap{
  print "key: "k "value: " v
}
```

Taking input

```
-----  
let name be input("Enter your name: ")  
print "Hello, "name // Hello, <your entry>
```

More on Maps

```
Map[key,value] m be {1:"a", 2:"b"}
```

```
m.put(3, "c")  
print m.size()  
print m[3] // 3  
           c
```

```
Map[k,v] m be {1: "a", 2: "b"}  
foreach k,v : m {  
    print k v  
}  
// new loop foreach works with arrays  
and maps both
```

Interoperability with java

```
-----
```

```
let n be 10  
float f be 3.5  
let msg be "Hello"
```

```
@java {  
    vars.put("z", (int) vars.get("n") + 5);  
    vars.put("pi", 3.14159);  
    vars.put("shout", ((String) vars.get("msg")).toUpperCase());  
}
```

```
print n // 10  
print f // 3.5  
print msg // Hello  
print z // 15 (int)  
print pi // 3.14159 (double, formatted)  
print shout // HELLO
```

```
let x be 5  
float y be 2.5  
let name be "Syed Ishaq"
```

```
@java {  
    double z = VarUtils.getNumber(vars, "x");
```

```
double p = Math.pow(VarUtils.getNumber(vars, "y"), 3);
String s = "Hello " + VarUtils.getString(vars, "name");

vars.put("z", z);
vars.put("p", p);
vars.put("greeting", s);
}

print z
print p
print greeting
```