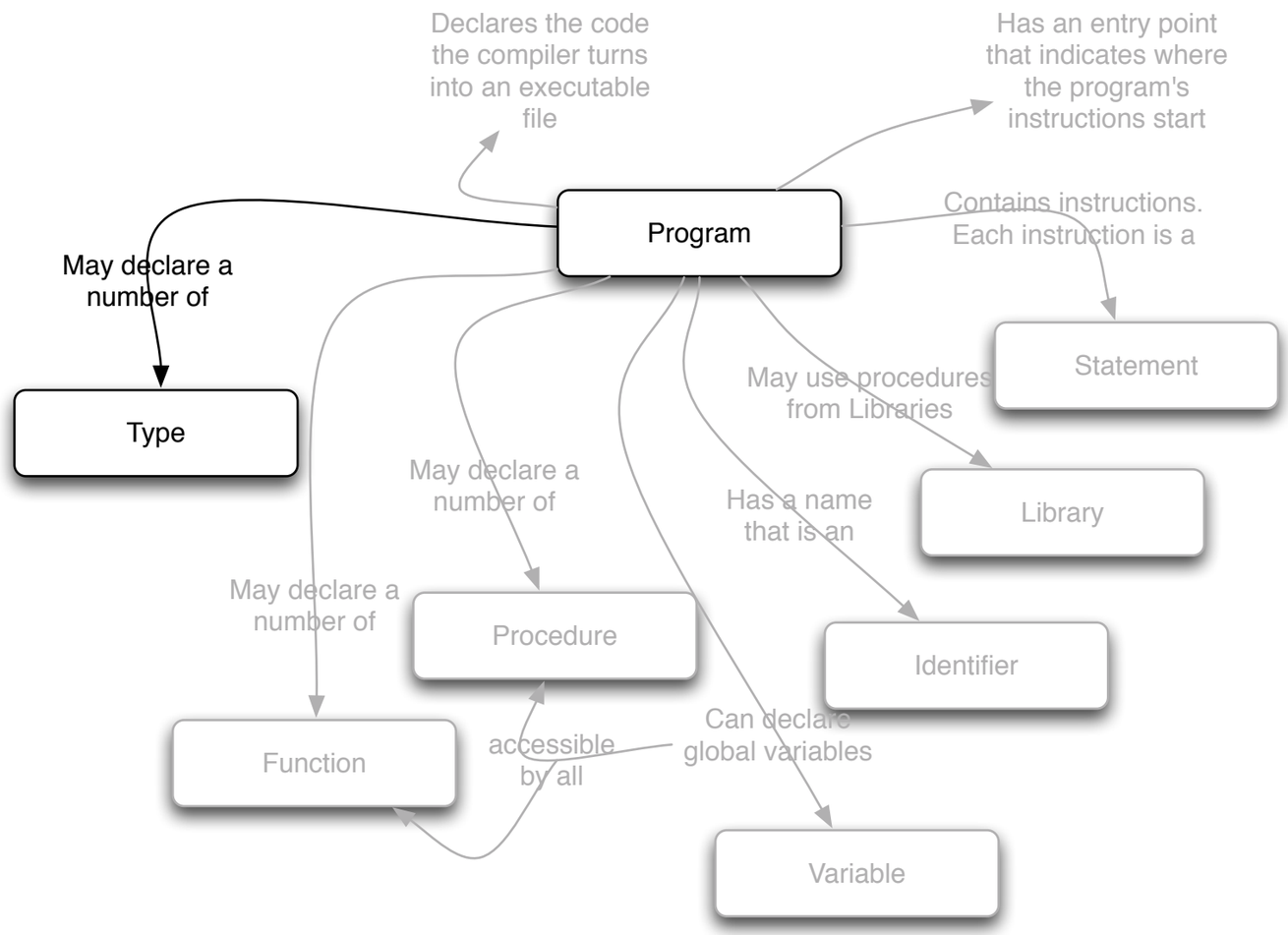
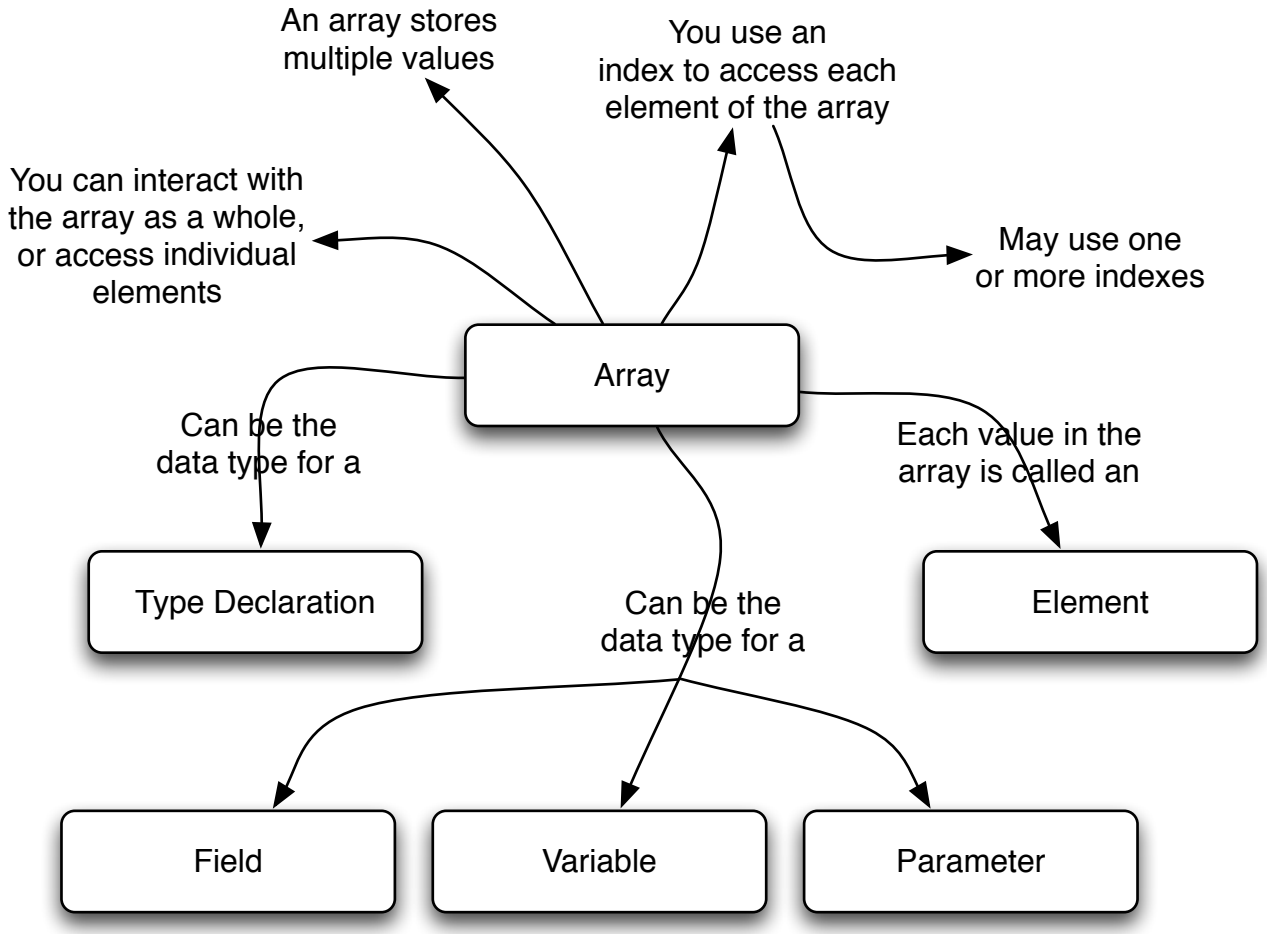
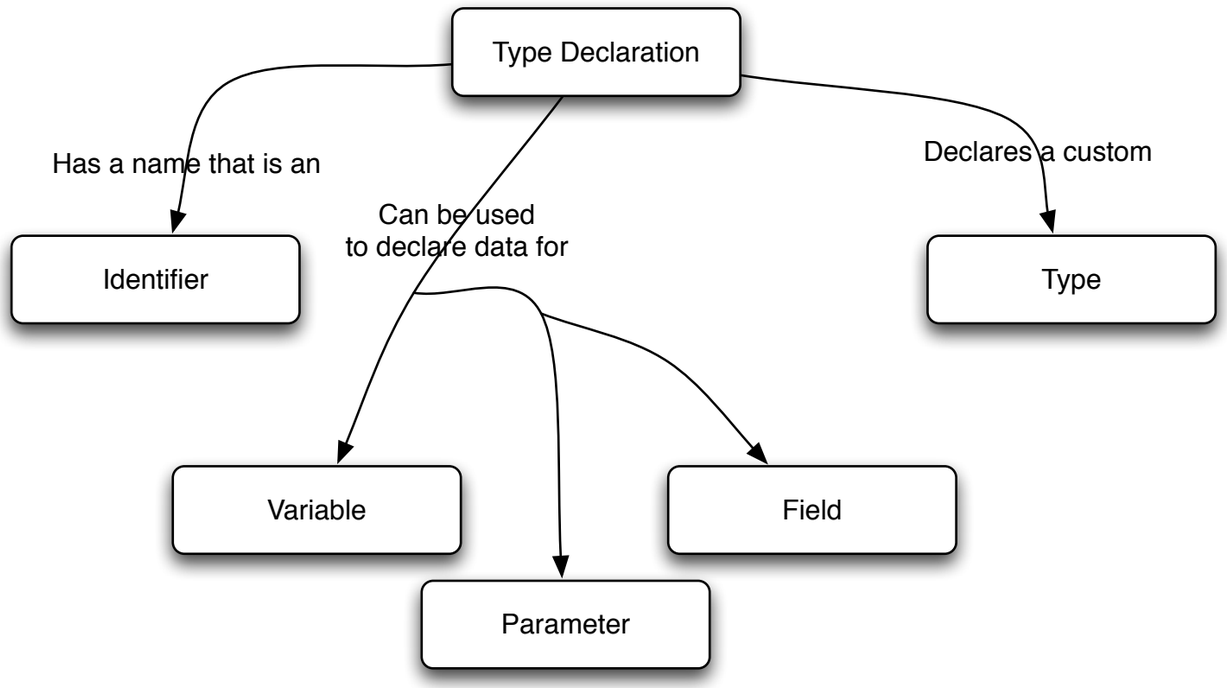
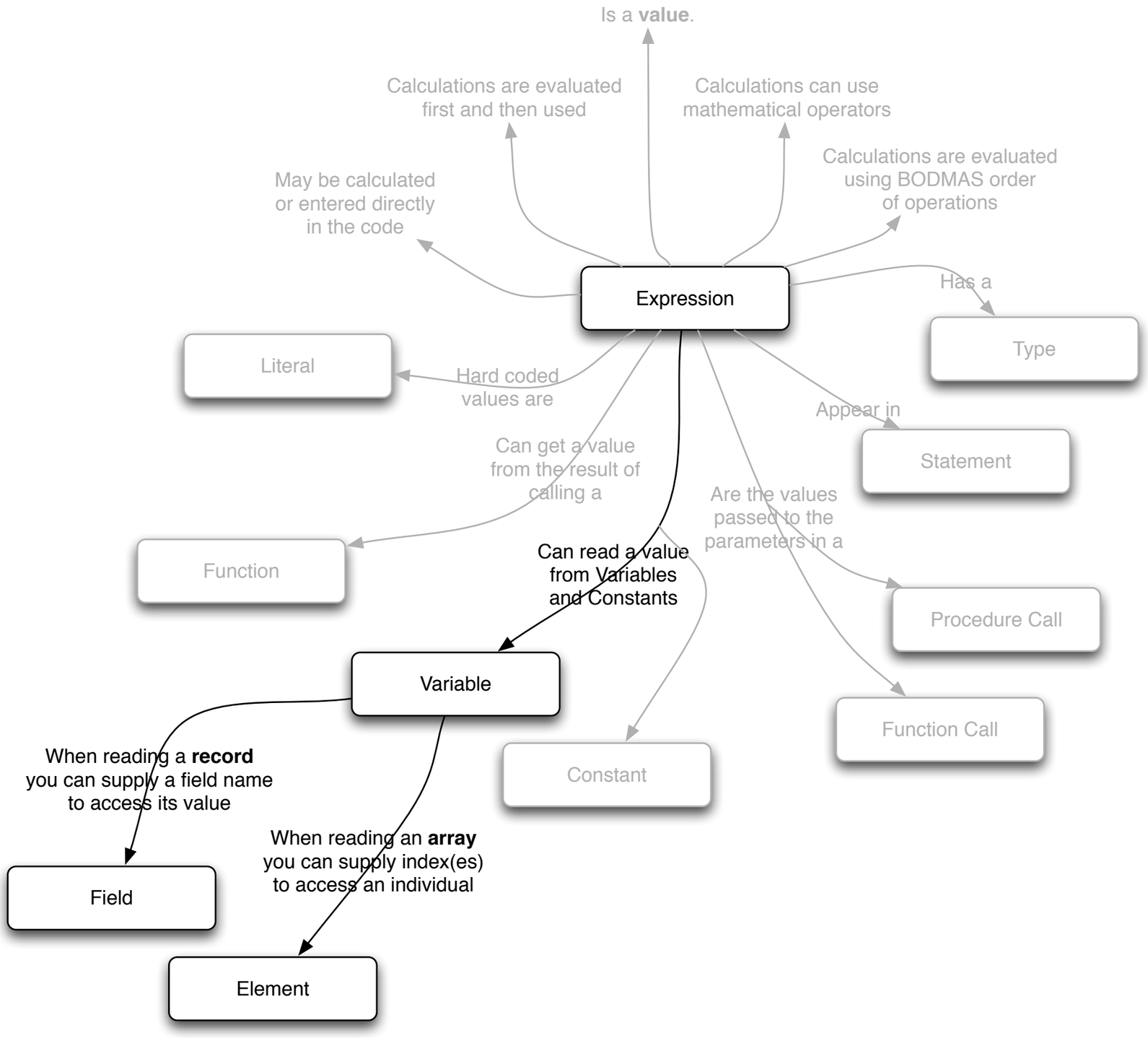
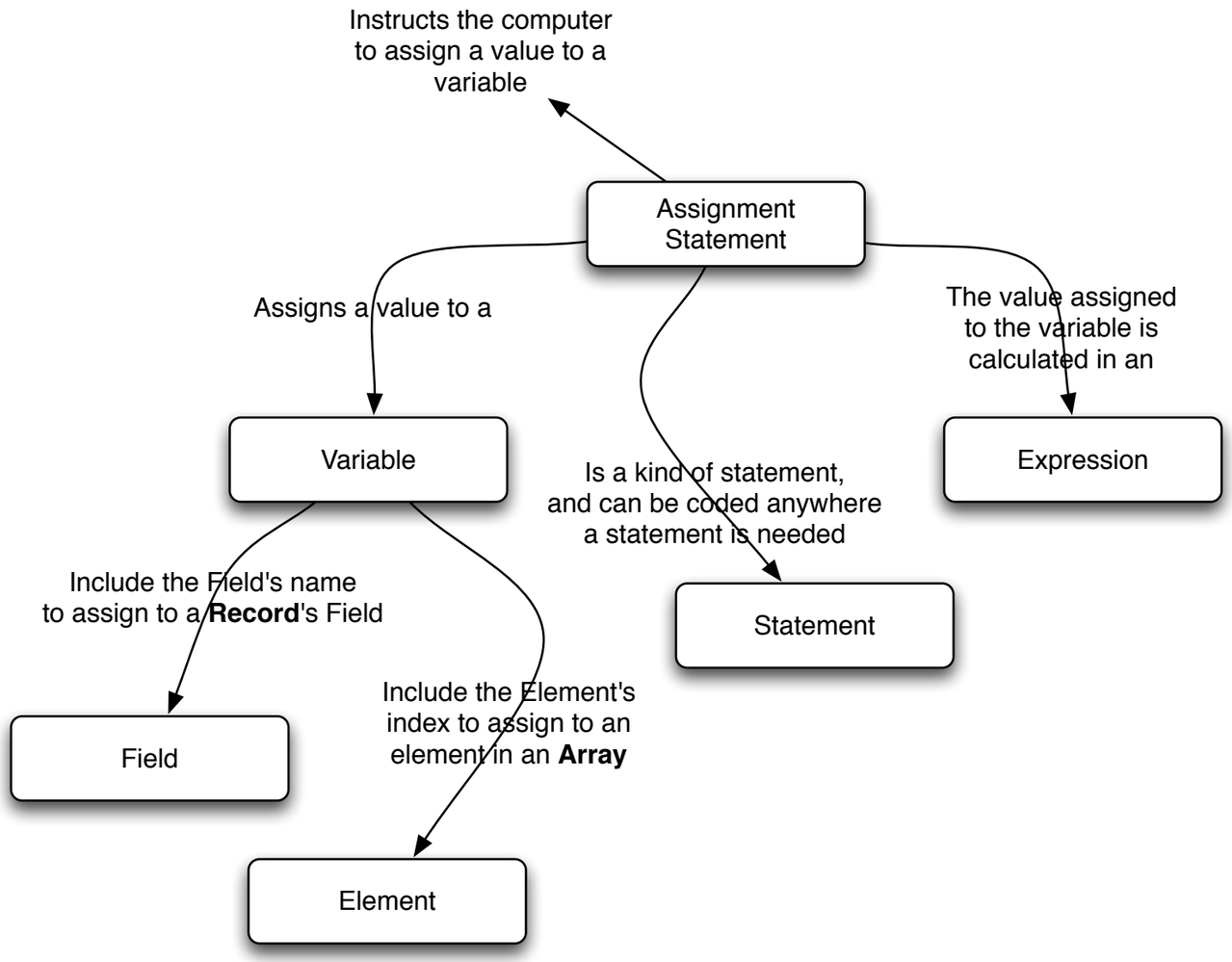
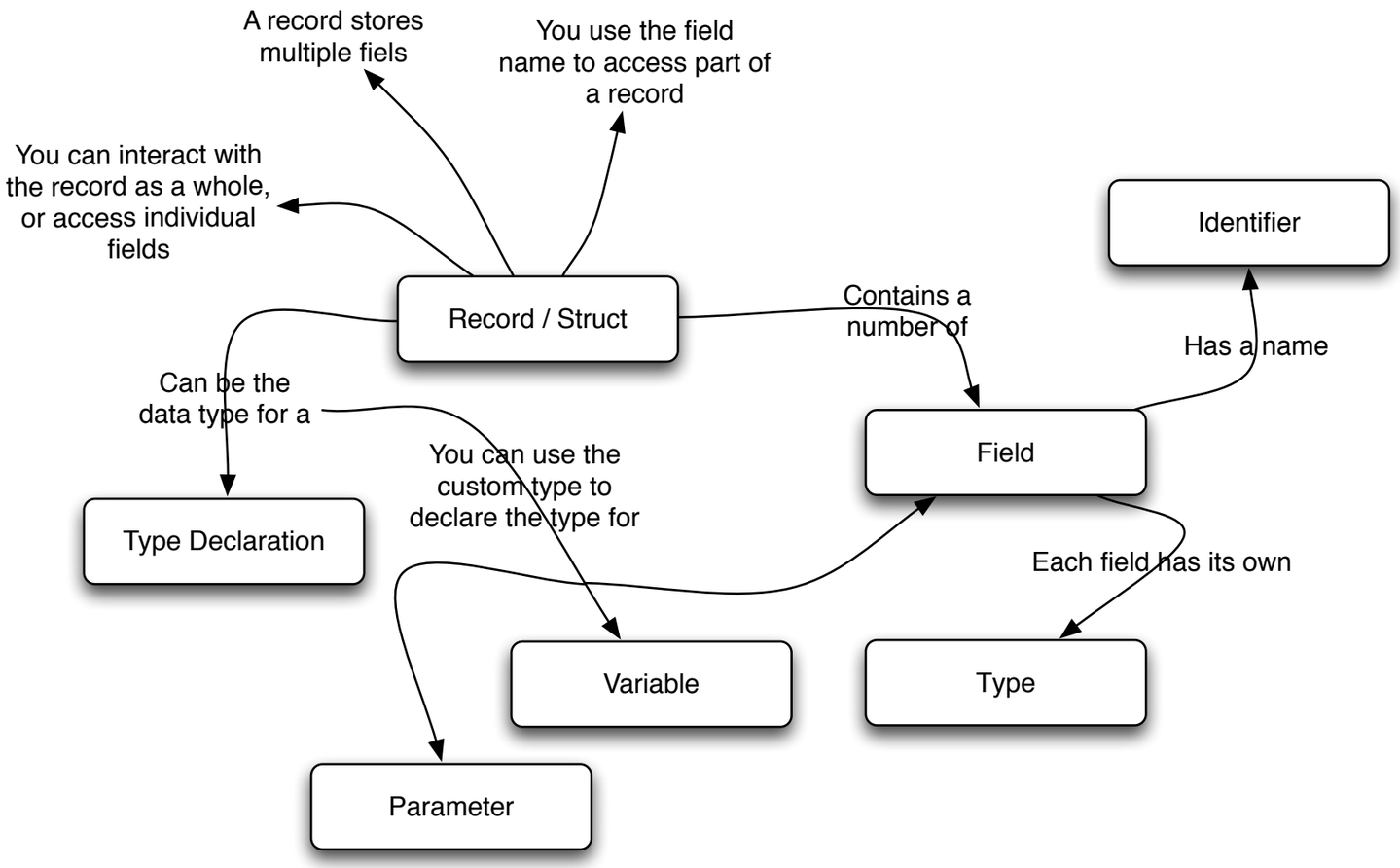

Declares the code the compiler turns into an executable file
Has an entry point that indicates where the program's instructions start
Program
Contains instructions. Each instruction is a
Statement
May declare a number of
Type
May use procedures from Libraries
Library
May declare a number of
Has a name that is an
Identifier
May declare a number of
Procedure
Function
accessible by all
Can declare global variables
Variable

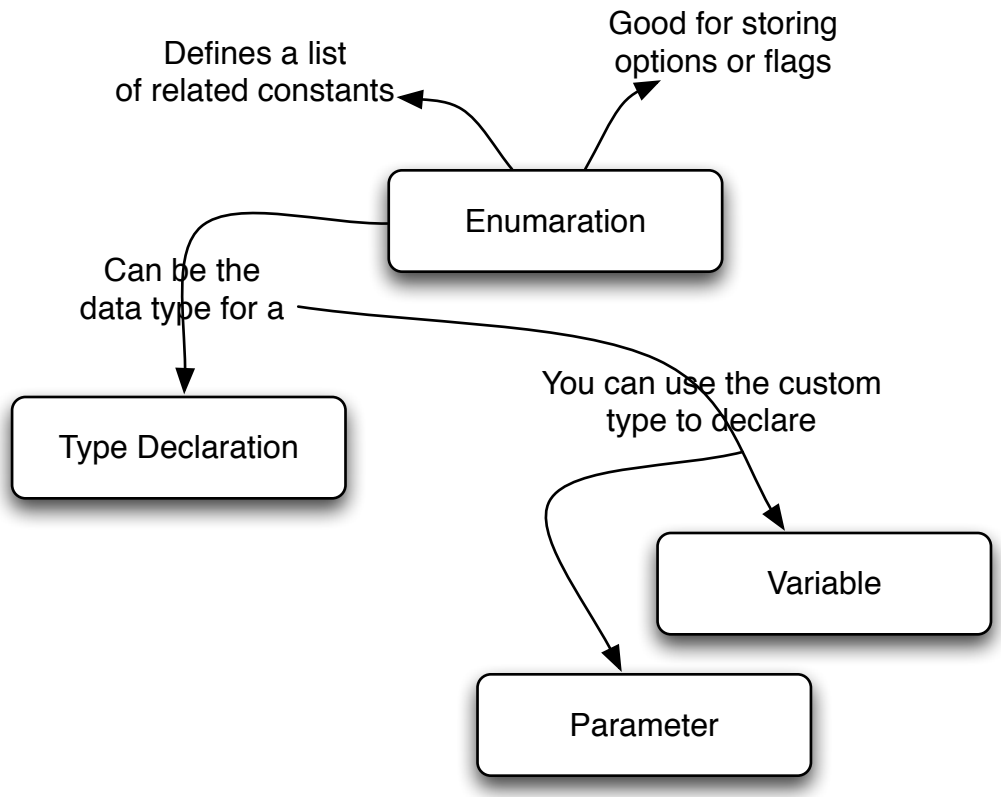

An array stores multiple values
You use an index to access each element of the array
You can interact with the array as a whole, or access individual elements
May use one or more indexes
Array
Can be the data type for a
Each value in the array is called an
Type Declaration
Element
Can be the data type for a
Field
Variable
Parameter

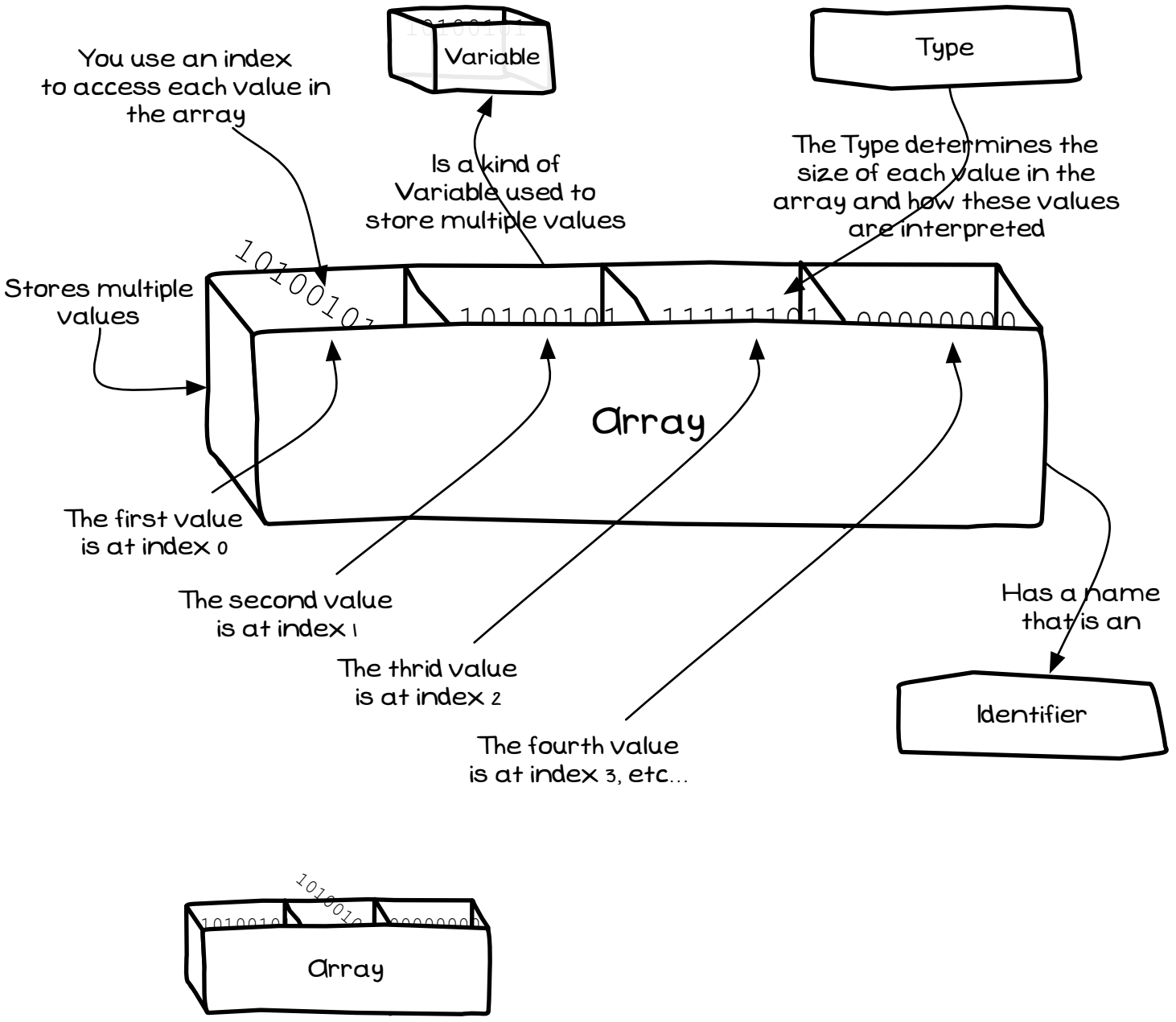

Type Declaration
Has a name that is an
Identifier
Can be used
to declare data for
Declares a custom
Type
Variable
Field
Parameter


Is a **value**.
Calculations are evaluated first and then used
Calculations can use mathematical operators
May be calculated or entered directly in the code
Calculations are evaluated using BODMAS order of operations
Expression
Has a
Type
Literal
Hard coded values are
Appear in
Statement
Can get a value from the result of calling a
Are the values passed to the parameters in a
Function
Can read a value from Variables and Constants
Procedure Call
Variable
Function Call
When reading a **record** you can supply a field name to access its value
Constant
When reading an **array** you can supply index(es) to access an individual
Field
Element

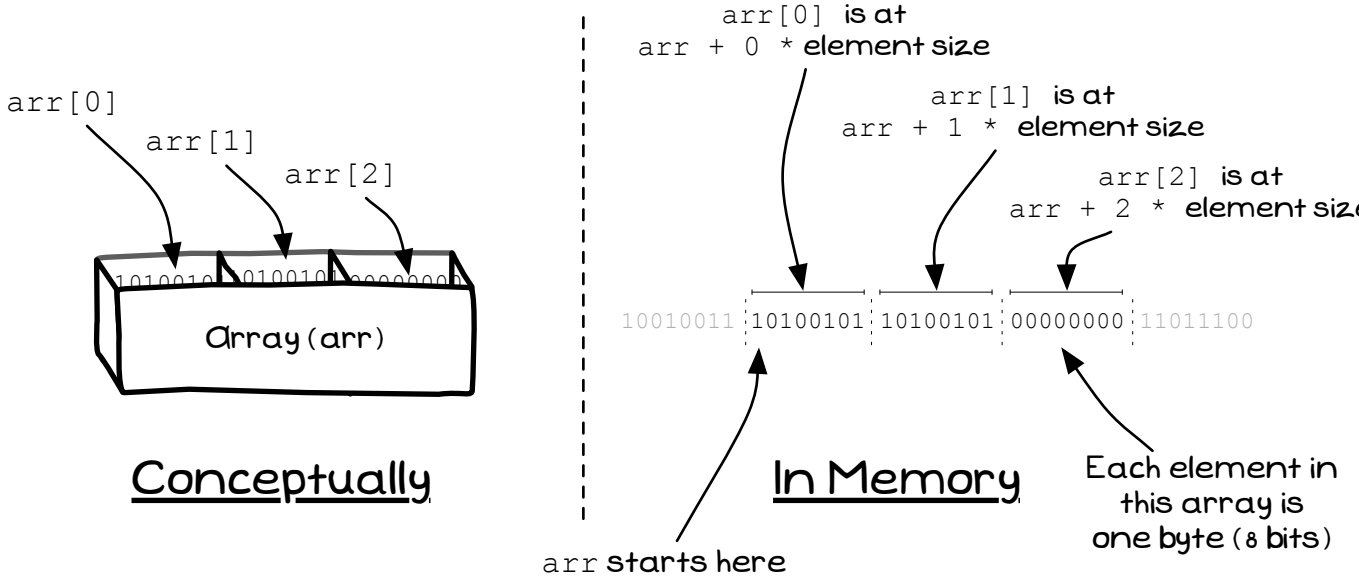

Instructs the computer to assign a value to a variable
Assignment Statement
Assigns a value to a
The value assigned to the variable is calculated in an
Variable
Expression
Is a kind of statement, and can be coded anywhere a statement is needed
Statement
Include the Field's name to assign to a **Record**'s Field
Include the Element's index to assign to an element in an **Array**
Field
Element

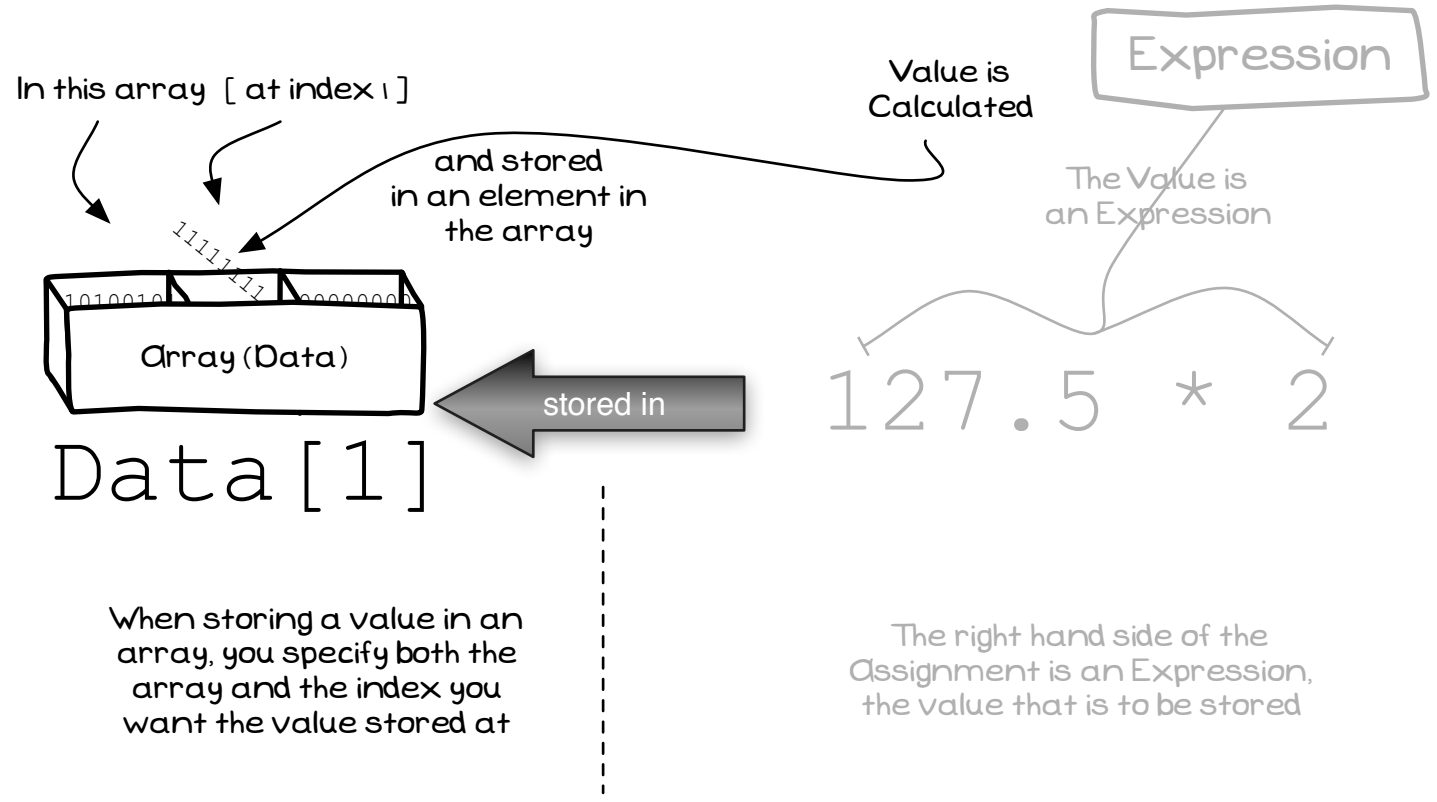

A record stores multiple fiels
You use the field name to access part of a record
You can interact with the record as a whole, or access individual fields
Record / Struct
Contains a number of
Identifier
Has a name
Field
Can be the data type for a
Type Declaration
You can use the custom type to declare the type for
Variable
Parameter
Each field has its own
Type


Defines a list
of related constants
Good for storing
options or flags
Enumaration
Can be the
data type for a
Type Declaration
You can use the custom
type to declare
Variable
Parameter


Variable
Type
You use an index to access each value in the array
Is a kind of Variable used to store multiple values
The Type determines the size of each value in the array and how these values are interpreted
Stores multiple values
Array
The first value is at index 0
The second value is at index 1
The thrid value is at index 2
The fourth value is at index 3, etc...
Has a name that is an
Identifier

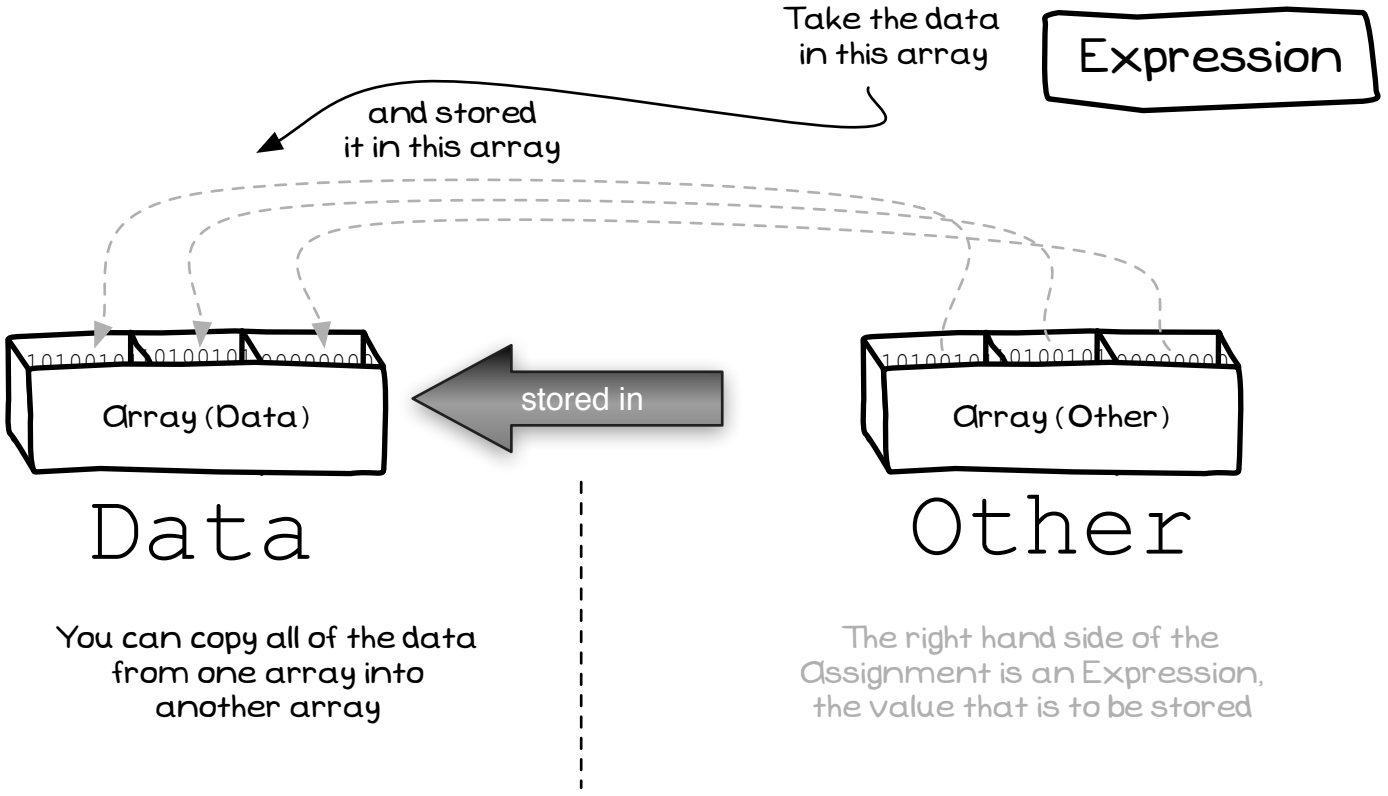
Array


arr[0]
arr[1]
arr[2]
Array (arr)
Conceptually
arr[0] is at
arr + 0 * element size
arr[1] is at
arr + 1 * element size
arr[2] is at
arr + 2 * element size
10010011
10100101
10100101
00000000
11011100
In Memory
arr starts here
Each element in
this array is
one byte (8 bits)



When storing a value in an array, you specify both the array and the index you want the value stored at

The right hand side of the Assignment is an Expression, the value that is to be stored

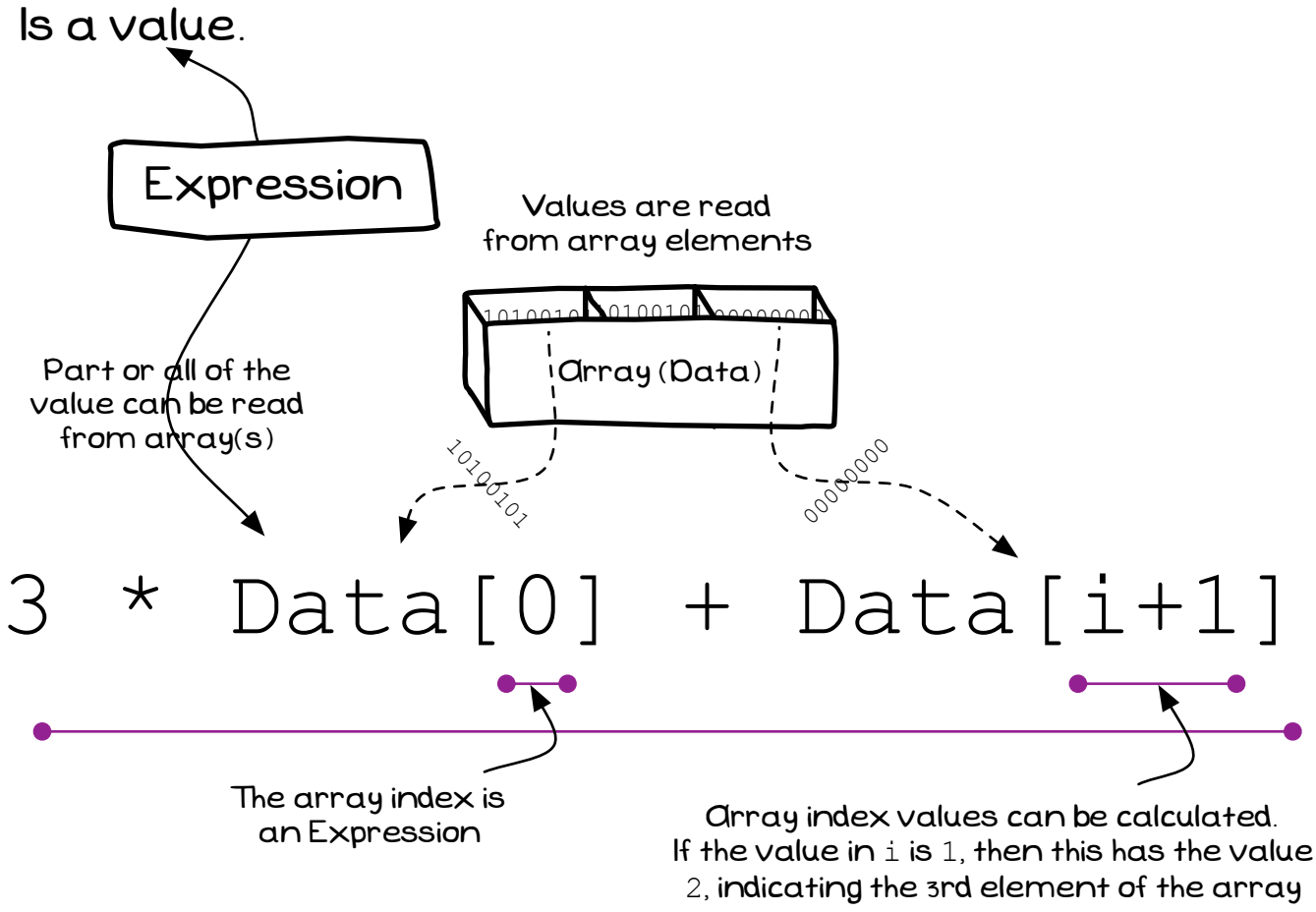


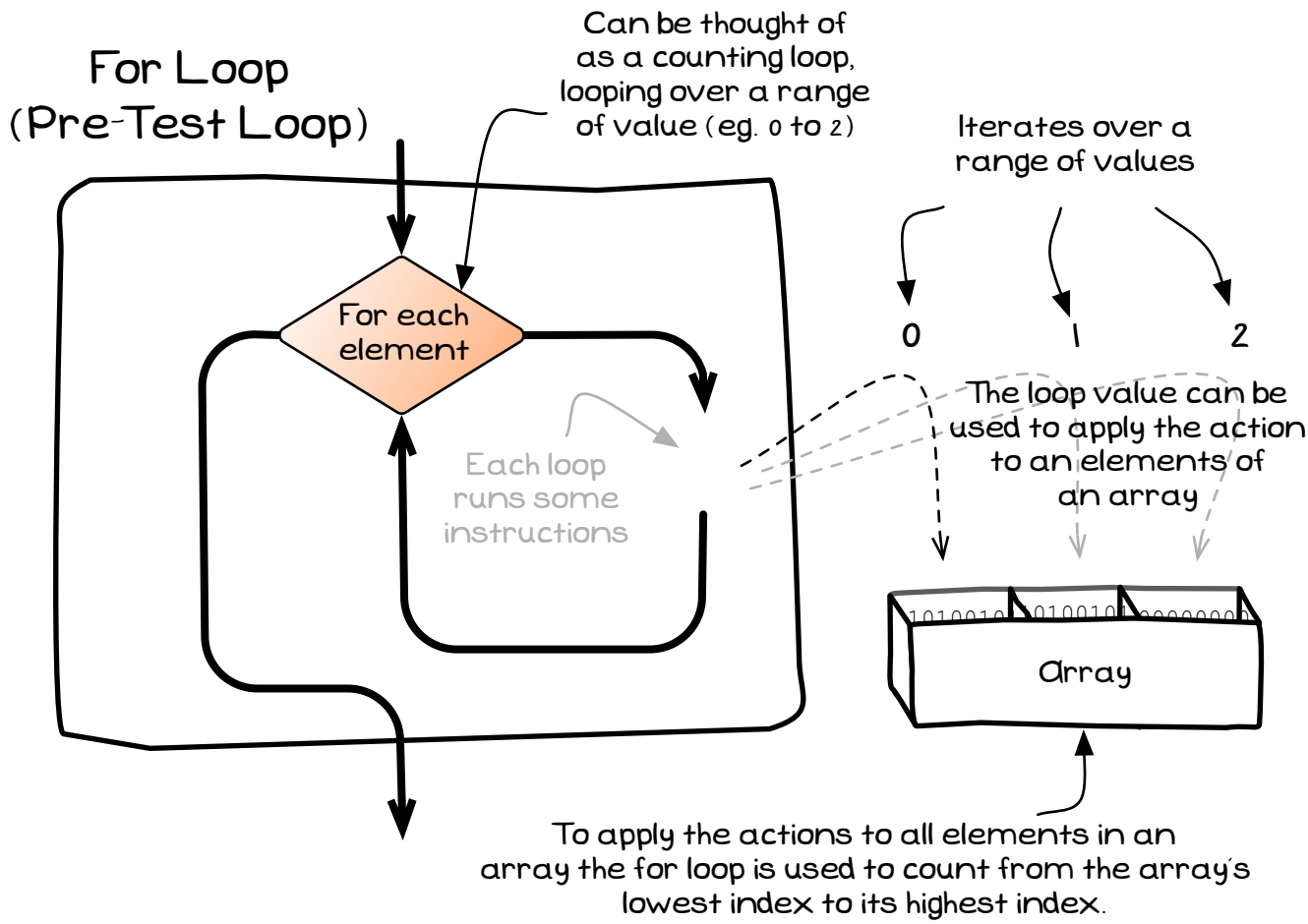
Data

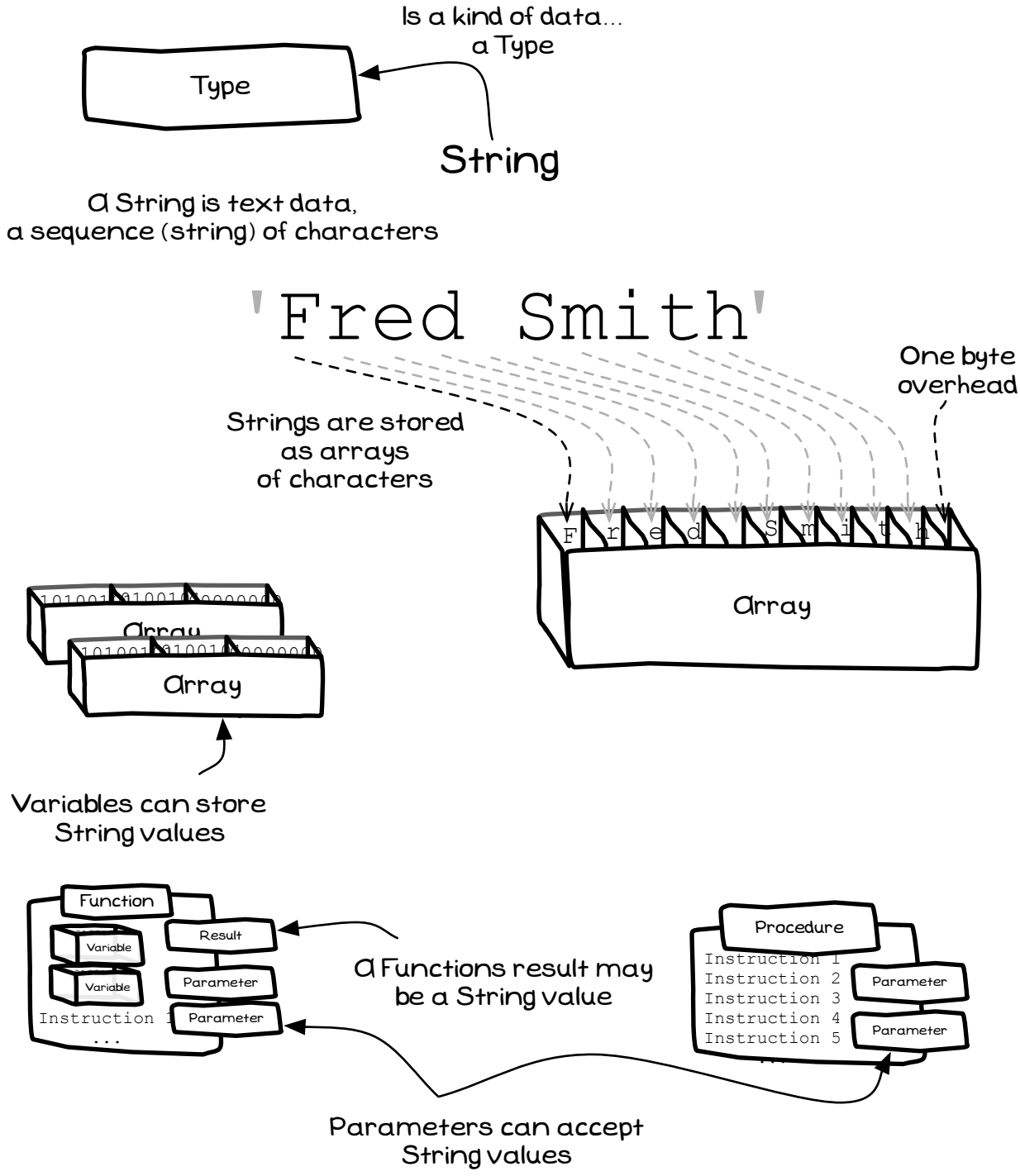
You can copy all of the data from one array into another array

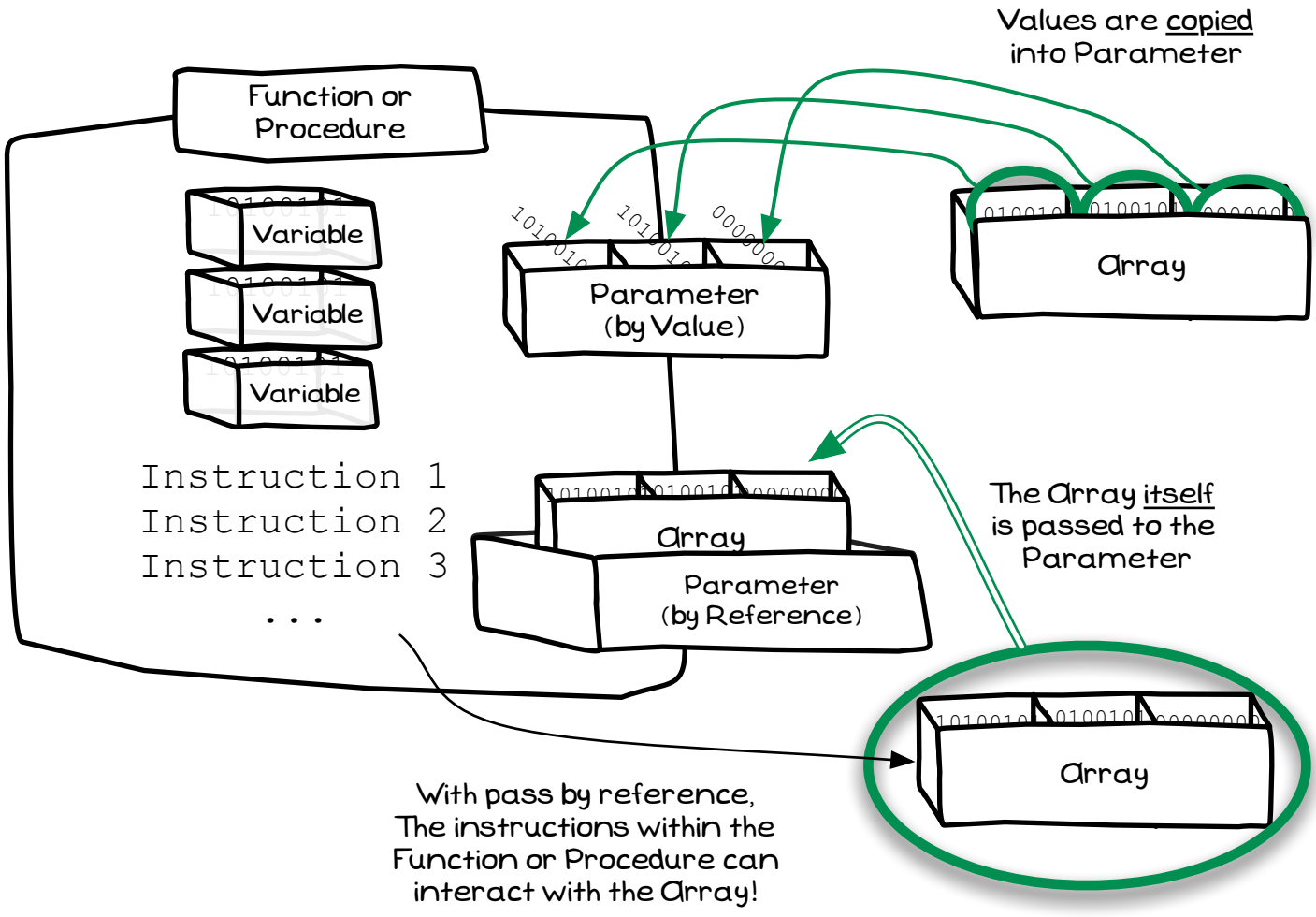
Other

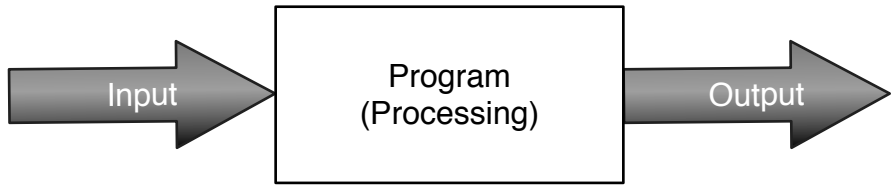
The right hand side of the Assignment is an Expression, the value that is to be stored

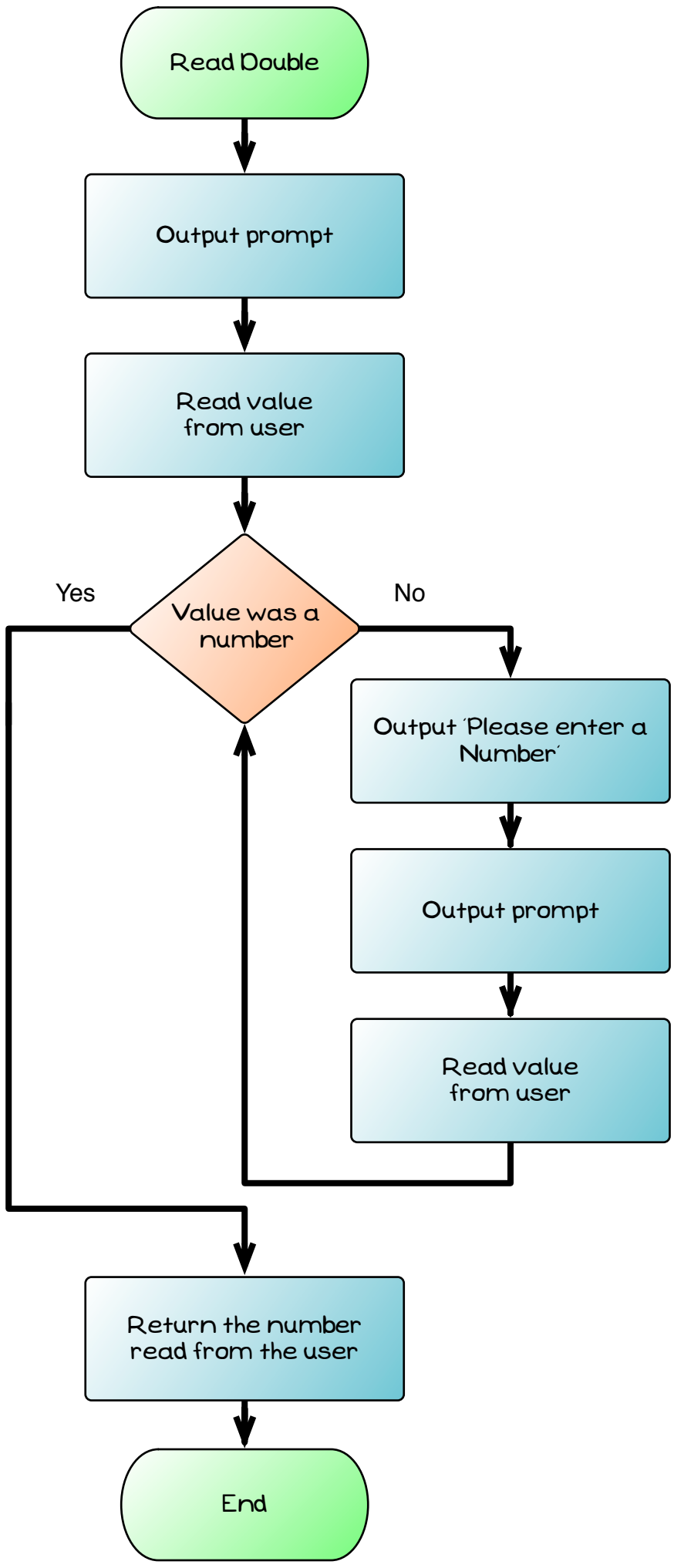

Is a value.
Expression
Values are read from array elements
Array (Data)
Part or all of the value can be read from array(s)
10100101
00000000
3 * Data[0] + Data[i+1]
The array index is an Expression
Array index values can be calculated. If the value in i is 1, then this has the value 2, indicating the 3rd element of the array

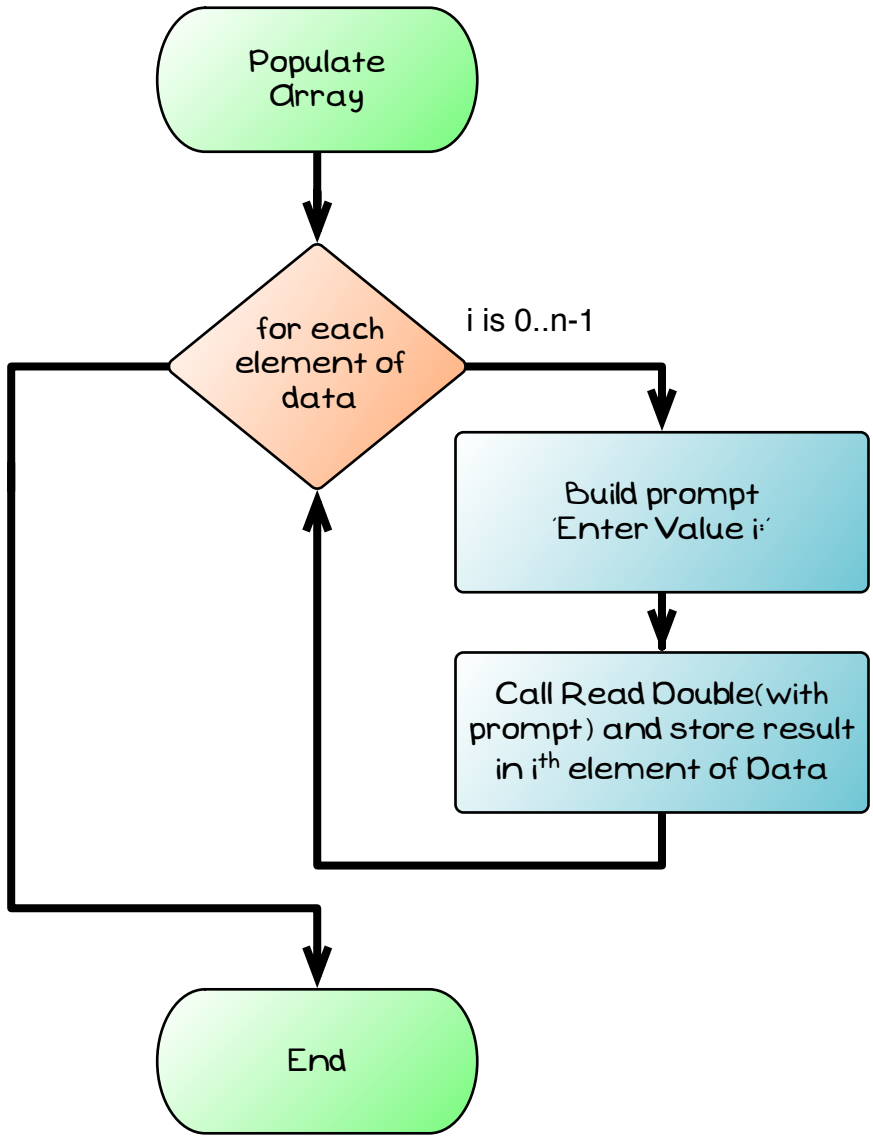

For Loop
(Pre-Test Loop)
Can be thought of as a counting loop, looping over a range of value (eg. 0 to 2)
Iterates over a range of values
0
1
2
For each element
Each loop runs some instructions
The loop value can be used to apply the action to an elements of an array
Array
To apply the actions to all elements in an array the for loop is used to count from the array's lowest index to its highest index.

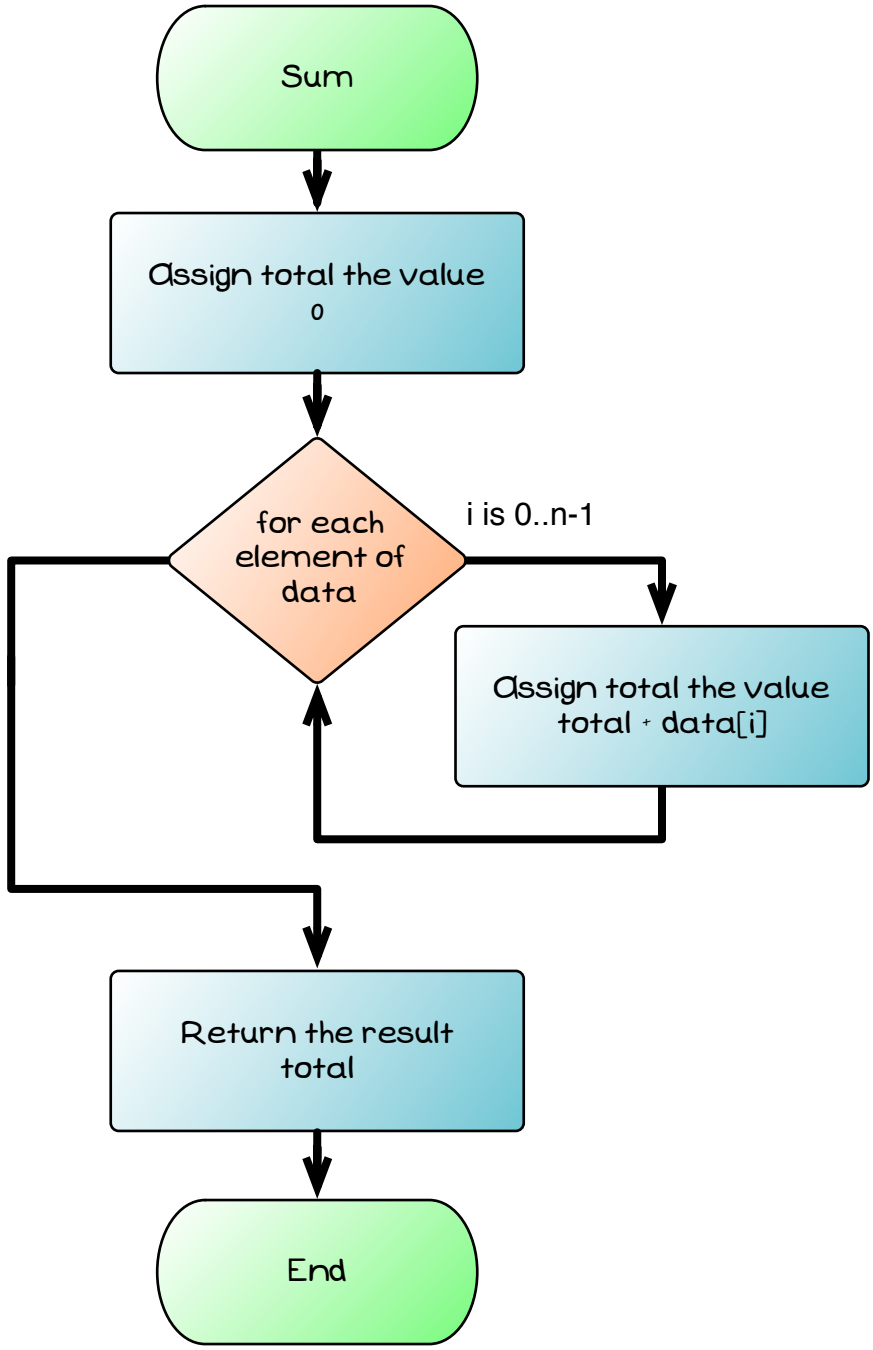

Is a kind of data...
a Type
Type
String
A String is text data,
a sequence (string) of characters
'Fred Smith'
One byte
overhead
Strings are stored
as arrays
of characters
F r e d S m i t h
Array
Array
Array
Variables can store
String values
Function
Variable
Variable
Result
Parameter
Parameter
Instruction 1
...
A Functions result may
be a String value
Procedure
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Parameter
Parameter
Parameters can accept
String values

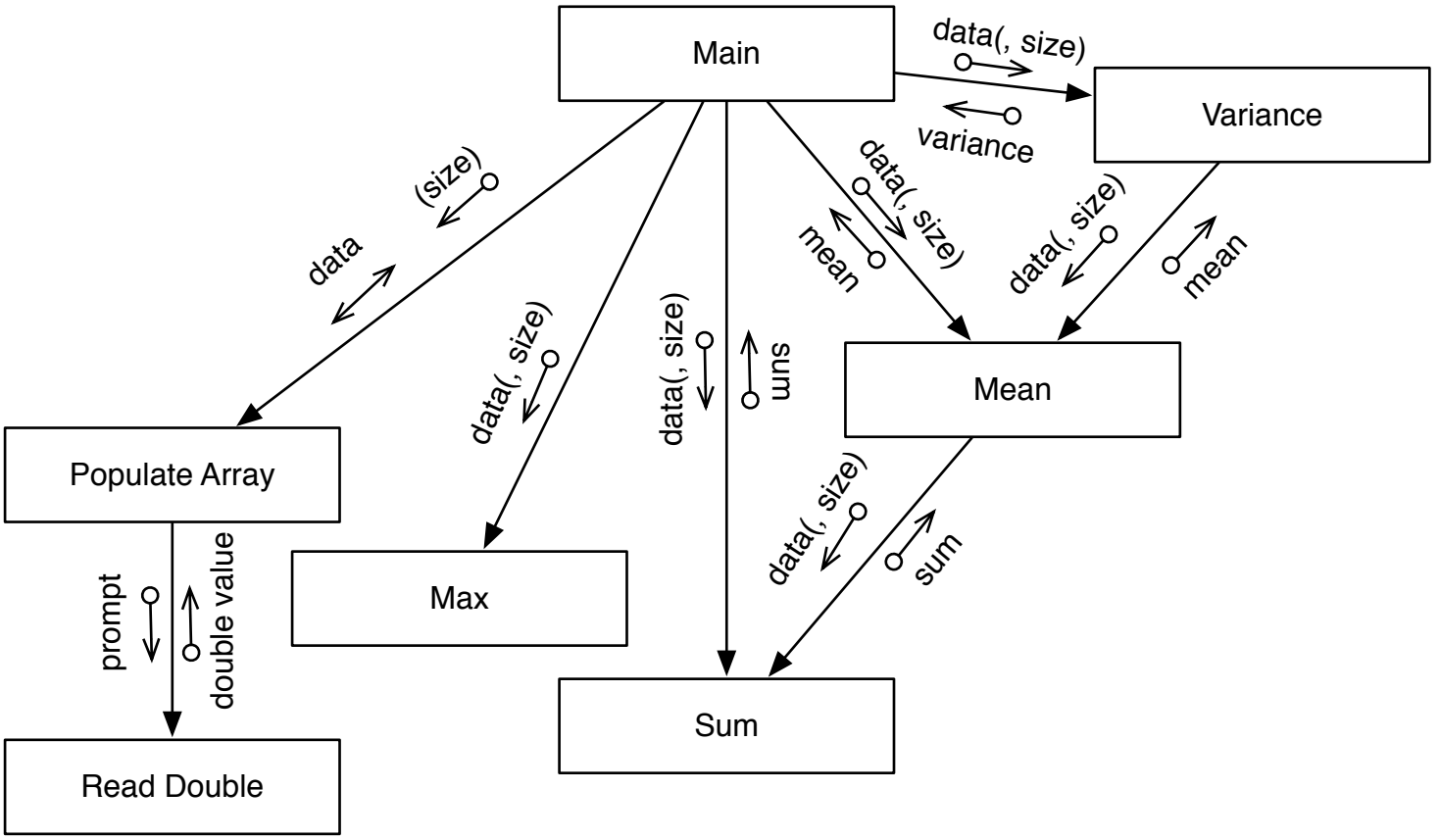

Values are copied into Parameter
Function or Procedure
Variable
Variable
Variable
Instruction 1
Instruction 2
Instruction 3
...
Parameter (by Value)
Array
Array
Parameter (by Reference)
The Array itself is passed to the Parameter
Array
With pass by reference, The instructions within the Function or Procedure can interact with the Array!

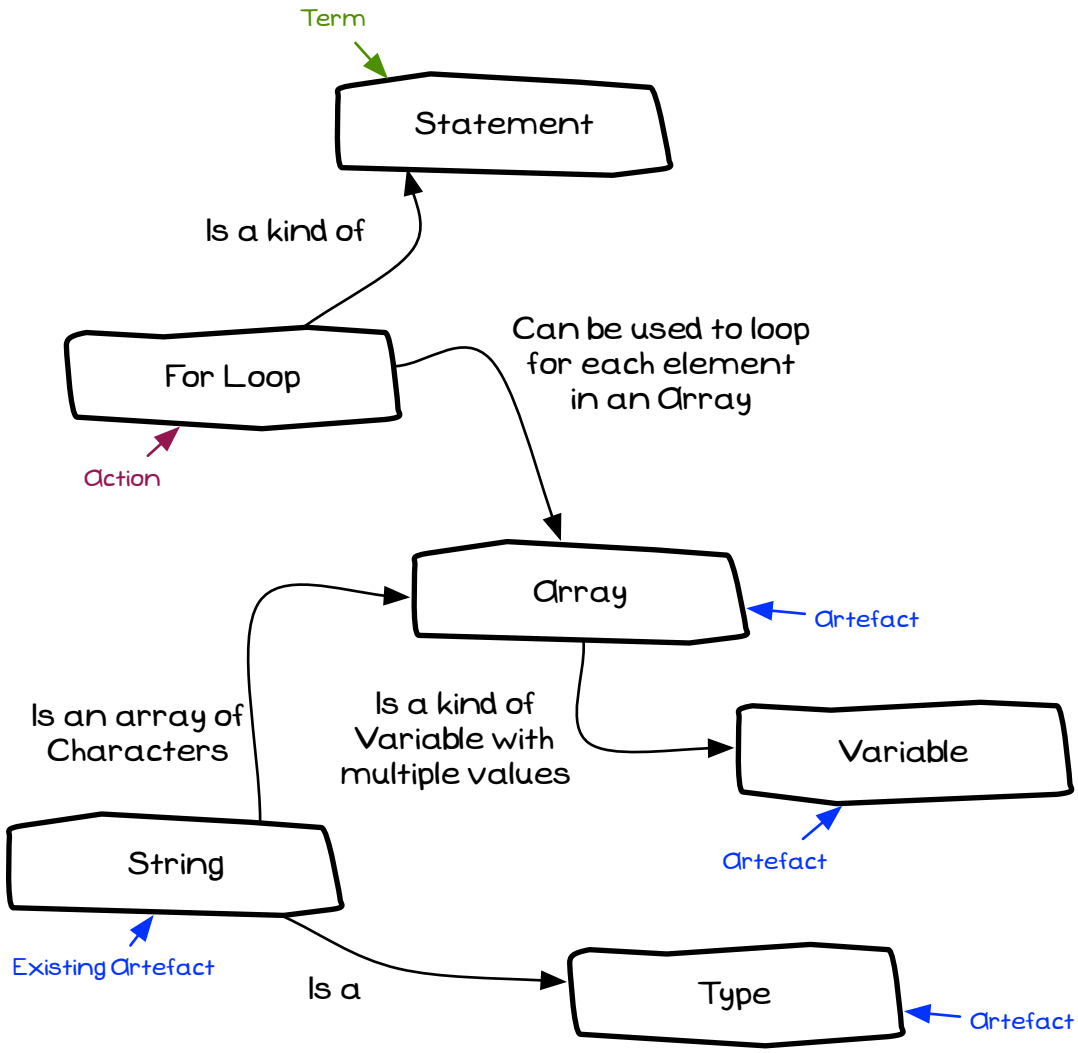

Input
Program
(Processing)
Output

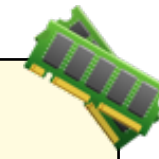

Read Double
Output prompt
Read value from user
Value was a number
Yes
No
Output 'Please enter a Number'
Output prompt
Read value from user
Return the number read from the user
End


Populate Array
for each element of data
i is 0..n-1
Build prompt 'Enter Value i:'
Call Read Double(with prompt) and store result in i^th element of Data
End


Sum
Assign total the value 0
for each element of data
i is 0..n-1
Assign total the value total + data[i]
Return the result total
End

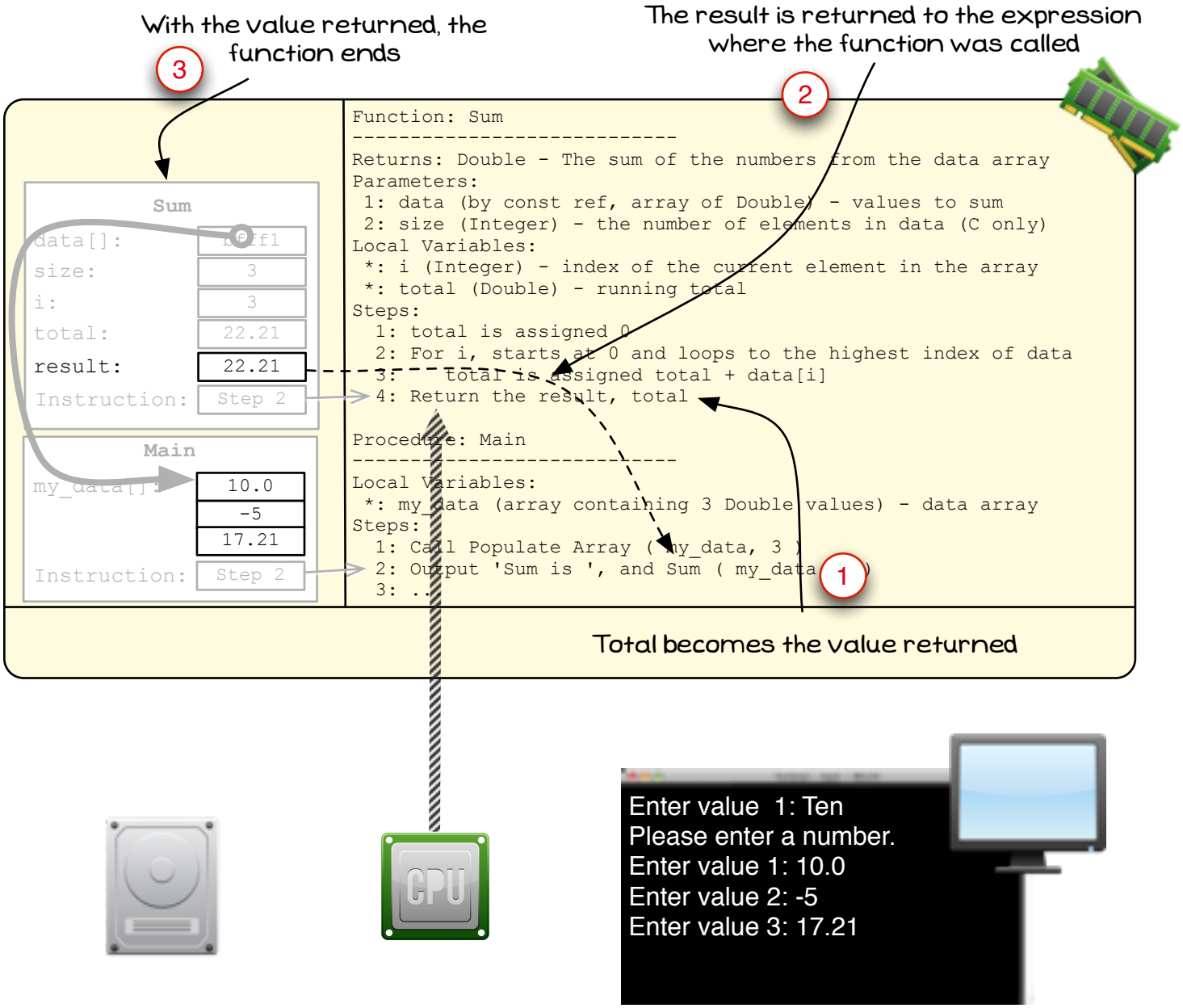

Main
data(, size)
variance
Variance
(size)
data
data(, size)
mean
data(, size)
mean
data(, size)
sum
Mean
Populate Array
data(, size)
Max
prompt
double value
data(, size)
sum
Sum
Read Double


Term
Statement
Is a kind of
For Loop
Can be used to loop for each element in an Array
Action
Array
Artefact
Is an array of Characters
Is a kind of Variable with multiple values
Variable
Artefact
String
Existing Artefact
Is a
Type
Artefact



```
Procedure: Populate Array
----------------------------
Parameters:
 1: data (by ref, array of Double) - the list of values to sum
 2: size (Integer) - the number of elements in data (C only)
Local Variables:
 *: i (Integer) - index of the current element in the array
 *: prompt (String) - the prompt to be shown to the user
Steps:
 1: For i, loops from lowest to highest index of data
 2:   Assign prompt, 'Enter value ' + (i + 1) + ': '
 3:   Assign data[i], the result of calling Read Double(prompt)

Procedure: Main
----------------------------
Local Variables:
 *: my_data (array containing 3 Double values) - data array
Steps:
  1: Call Populate Array ( my_data, 3 )
  2: ...
```


With the value returned, the function ends
3
The result is returned to the expression where the function was called
2
Sum
data[]:
size: 3
i: 3
total: 22.21
result: 22.21
Instruction: Step 2
Main
my_data[]:
10.0
-5
17.21
Instruction: Step 2
Function: Sum
Returns: Double - The sum of the numbers from the data array
Parameters:
1: data (by const ref, array of Double) - values to sum
2: size (Integer) - the number of elements in data (C only)
Local Variables:
*: i (Integer) - index of the current element in the array
*: total (Double) - running total
Steps:
1: total is assigned 0
2: For i, starts at 0 and loops to the highest index of data
3: total is assigned total + data[i]
4: Return the result, total
Procedure: Main
Local Variables:
*: my_data (array containing 3 Double values) - data array
Steps:
1: Call Populate Array (my_data, 3)
2: Output 'Sum is ', and Sum (my_data)
3: ..
1
Total becomes the value returned
CPU
Enter value 1: Ten
Please enter a number.
Enter value 1: 10.0
Enter value 2: -5
Enter value 3: 17.21

Before	Call	After

prompt:
`a s h e 2 8 & 2 k k ; ) j 2 ; 0 J`
buffer:
`# d Z`
	`strncpy(prompt, "Enter value ", 13);`	prompt:
		`E n t e r  v a l u e  ▫ 2 ; 0 J`
		buffer:
		`# d Z`

prompt:
`E n t e r  v a l u e  ▫ 2 ; 0 J`
buffer:
`# d Z`
	`sprintf(buffer, "%d", (i + 1) % 100);`	prompt:
		`E n t e r  v a l u e  ▫ 2 ; 0 J`
		buffer:
		`1 ▫ Z`

prompt:
`E n t e r  v a l u e  ▫ 2 ; 0 J`
buffer:
`1 ▫ Z`
	`strncat(prompt, buffer, 2);`	prompt:
		`E n t e r  v a l u e  1 ▫ ; 0 J`
		buffer:
		`1 ▫ Z`

prompt:
`E n t e r  v a l u e  1 ▫ ; 0 J`
buffer:
`1 ▫ Z`
	`strncat(prompt, ": ", 2);`	prompt:
		`E n t e r  v a l u e  1 :  ▫ J`
		buffer:
		`1 ▫ Z`