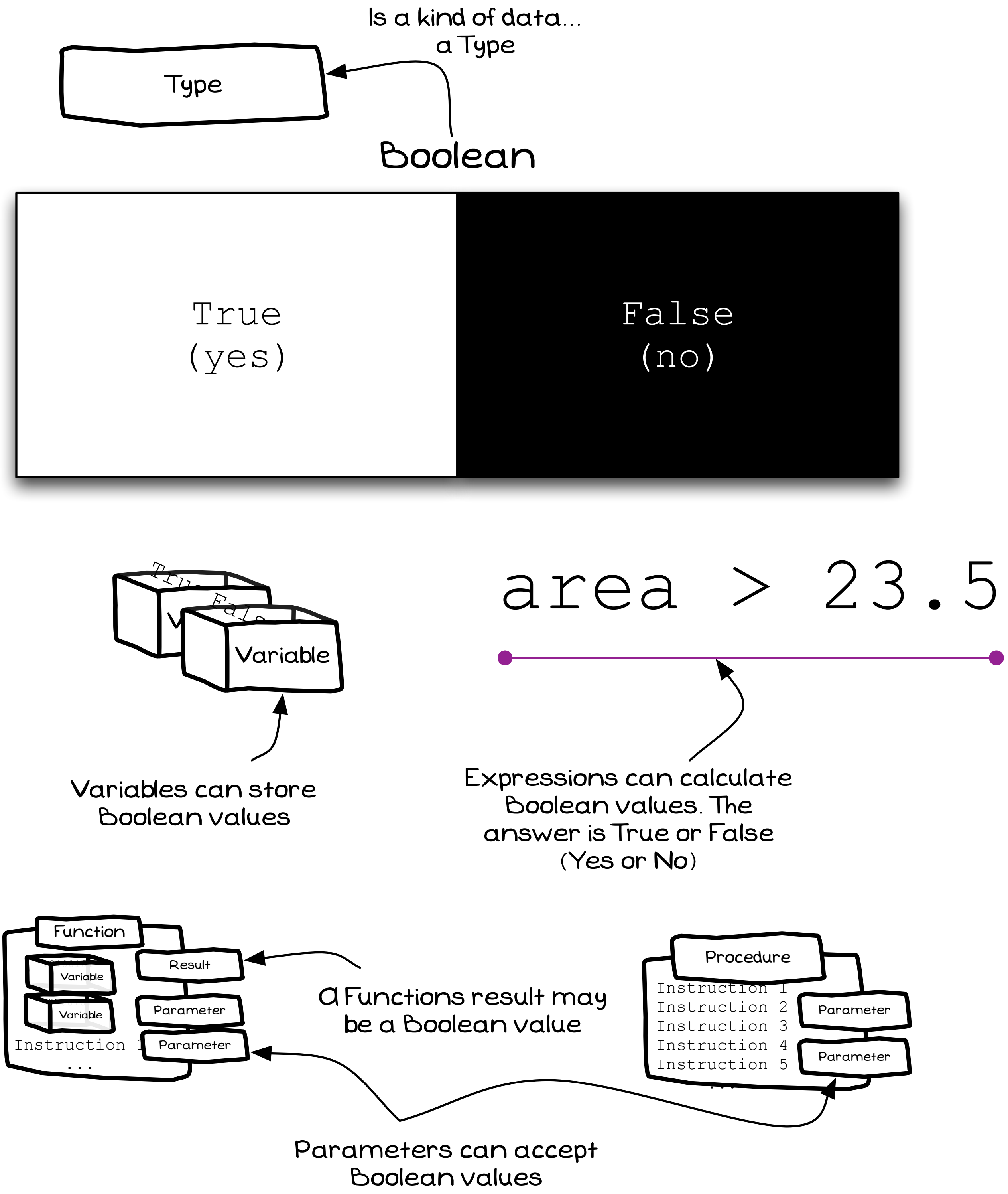


Is a kind of data...
a Type

Boolean

Variables can store
Boolean values

Expressions can calculate
Boolean values. The
answer is True or False
(Yes or No)

a Functions result may
be a Boolean value

Parameters can accept
Boolean values

Logical Operators

		b	
		True	False
a	True	True	False
	False	False	False

a **and** b

		b	
		True	False
a	True	True	True
	False	True	False

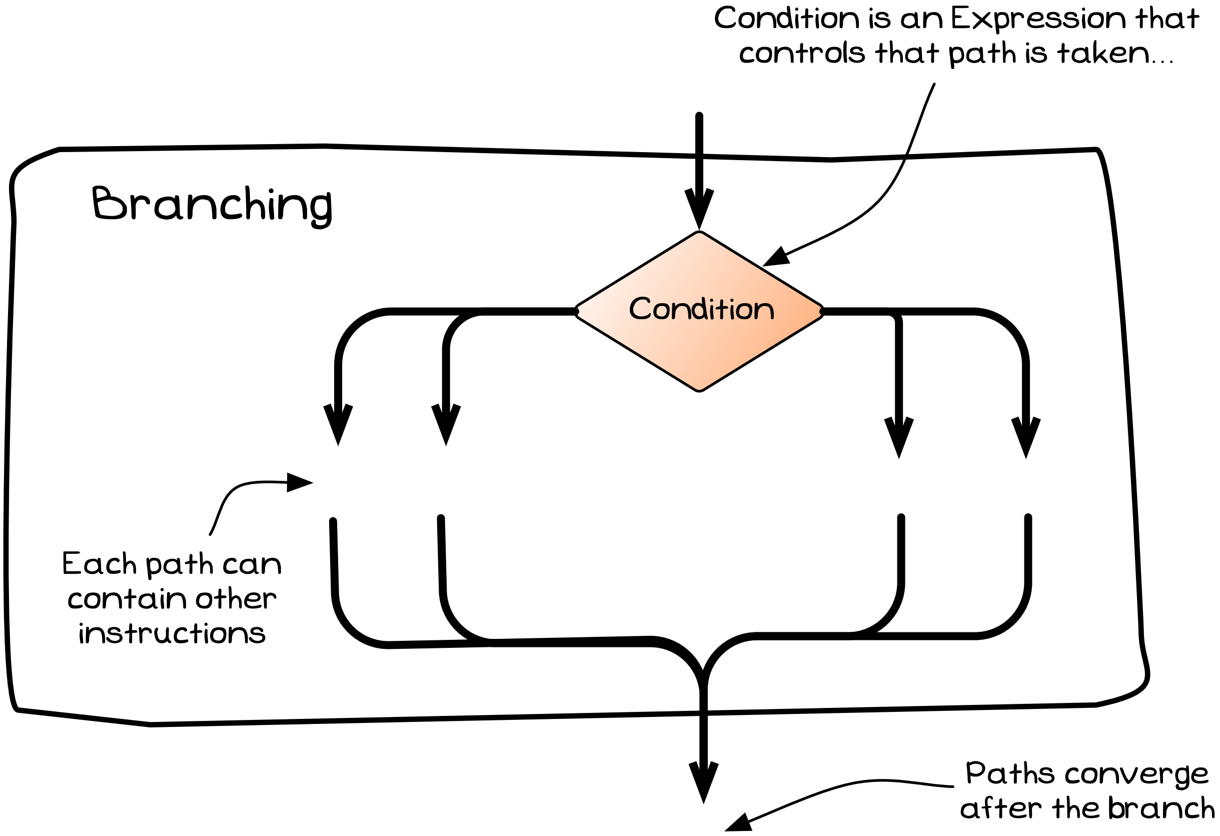
a **or** b

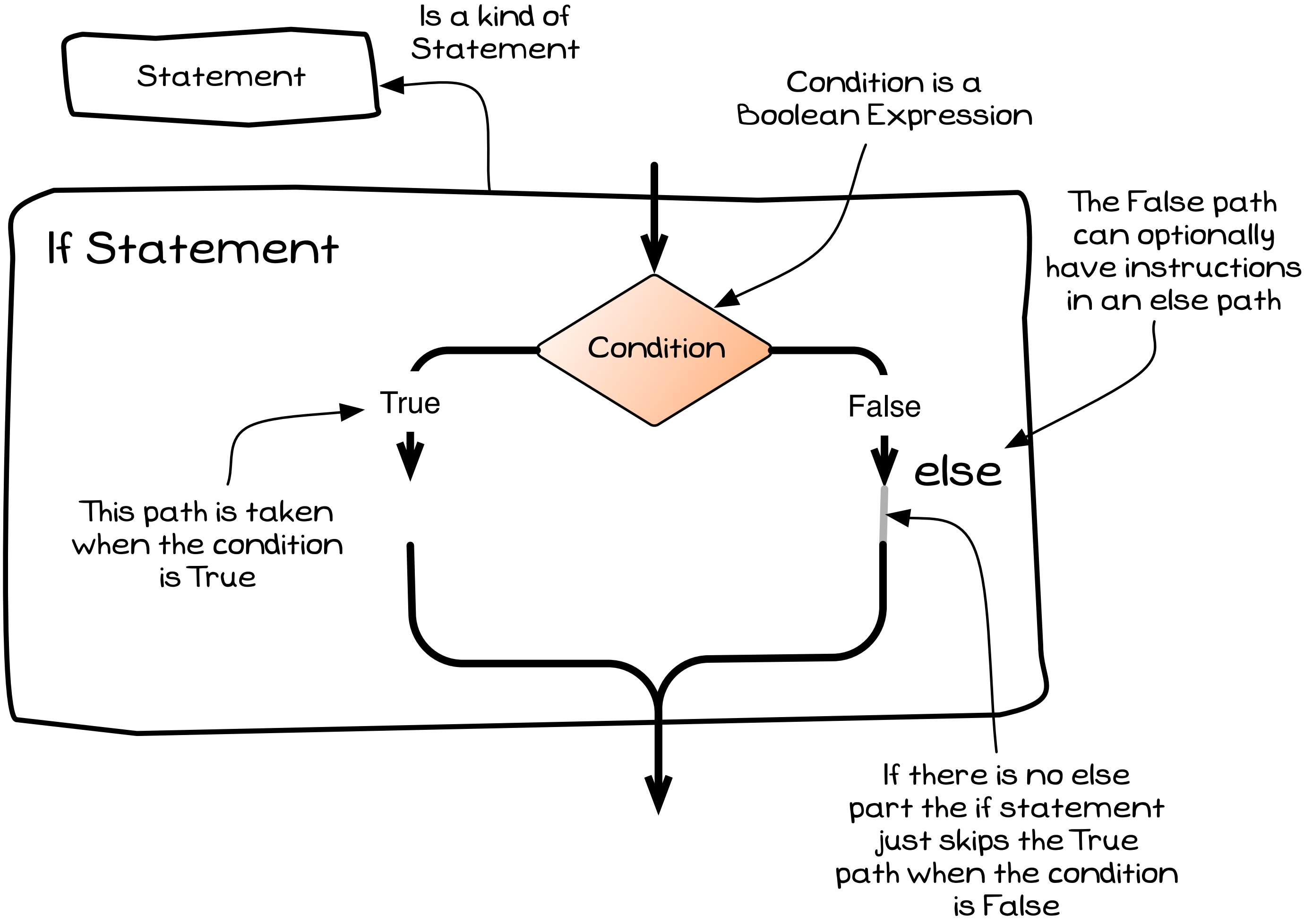
		b	
		True	False
a	True	False	True
	False	True	False

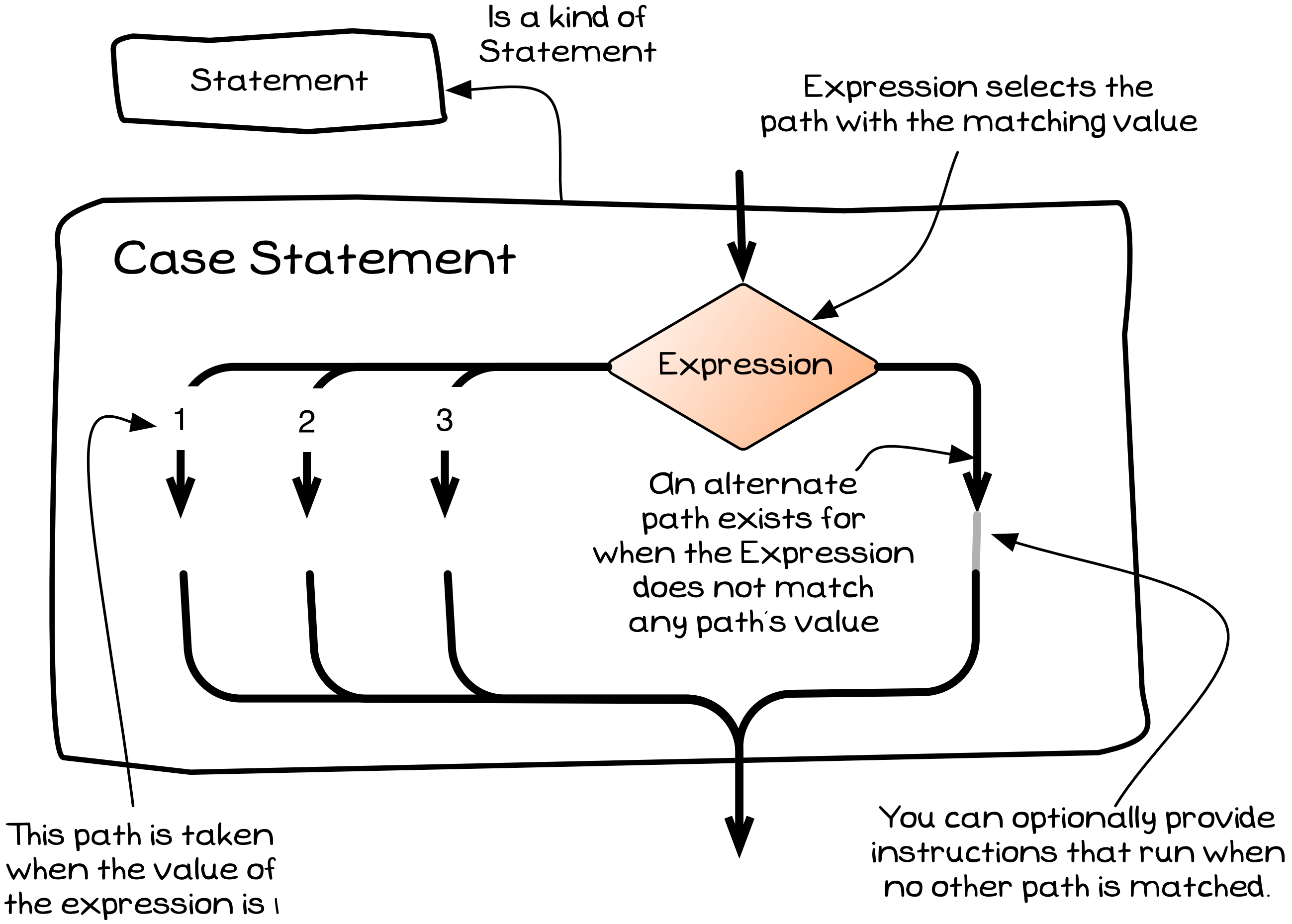
a **xor** b

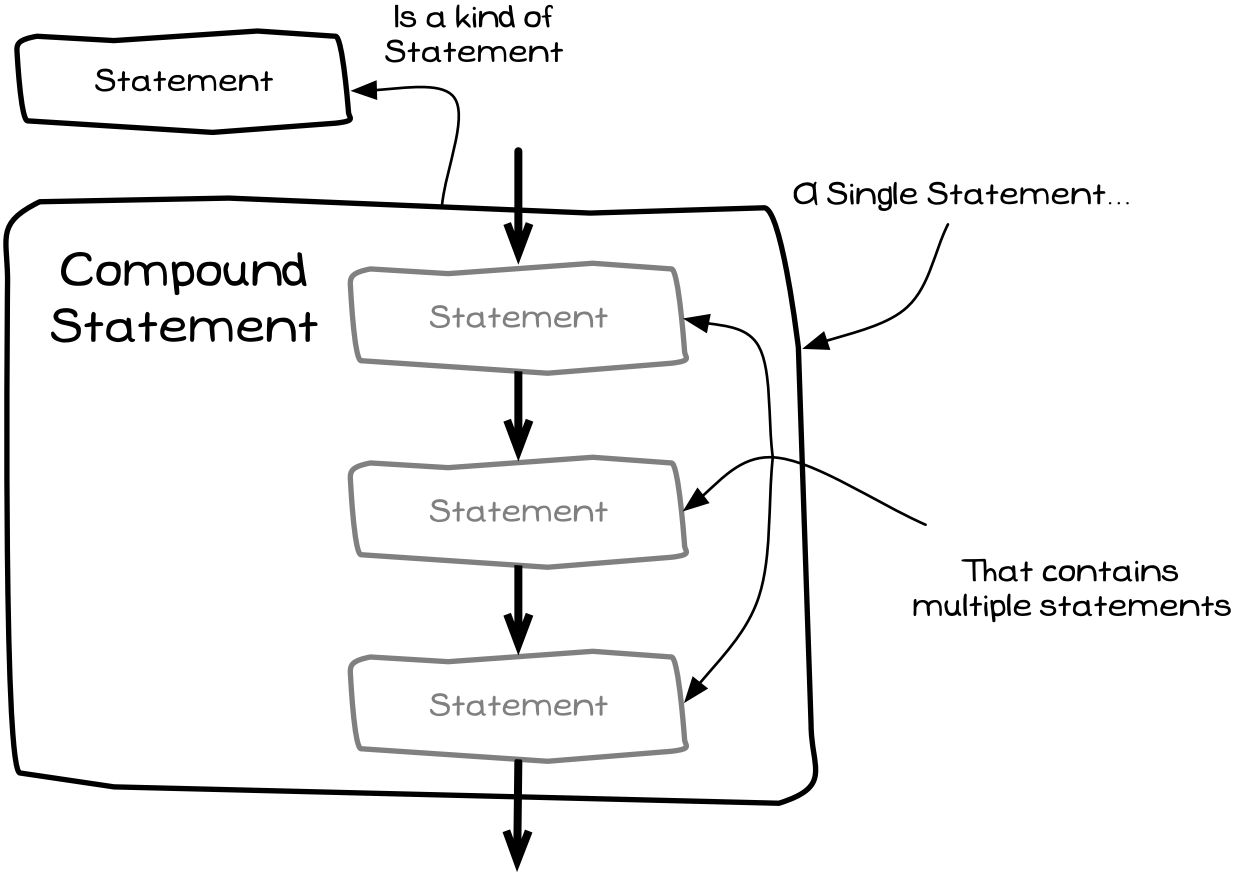
a	True	False
	False	True

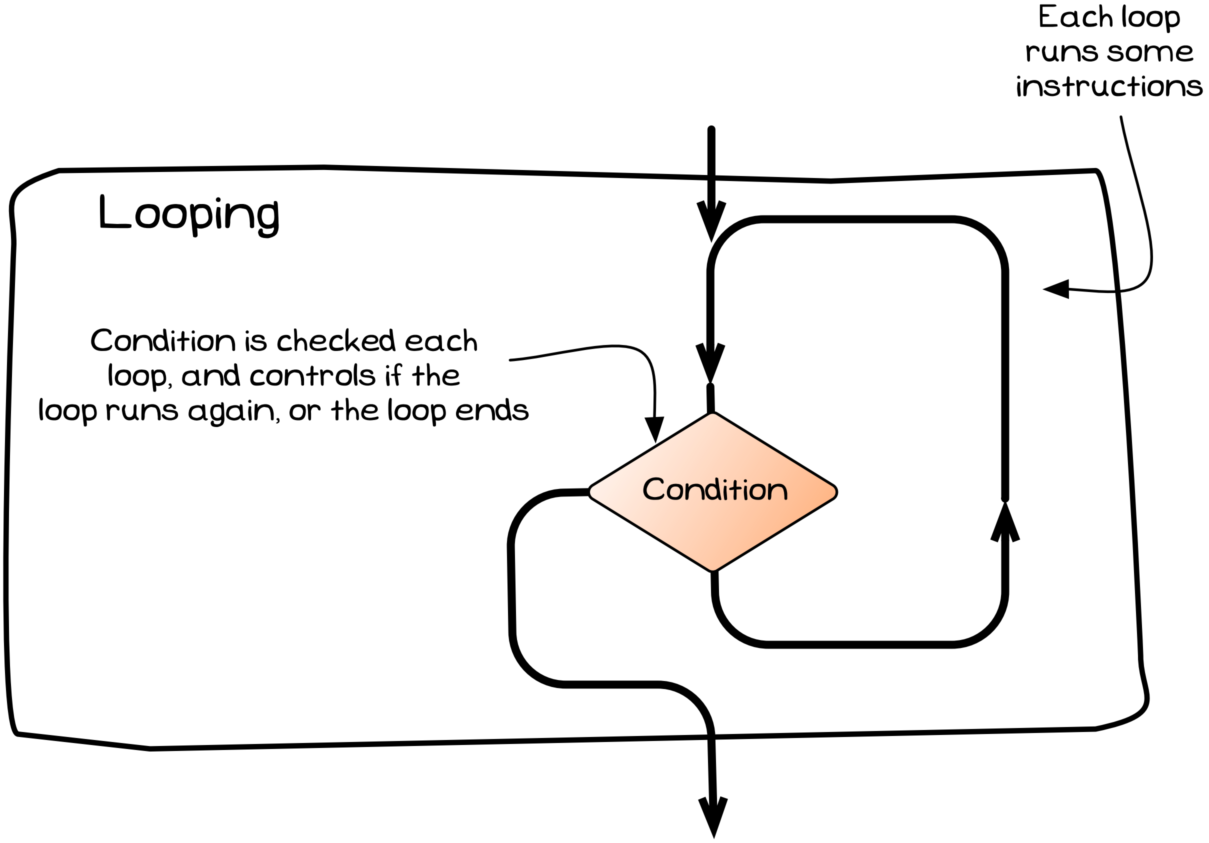
not a

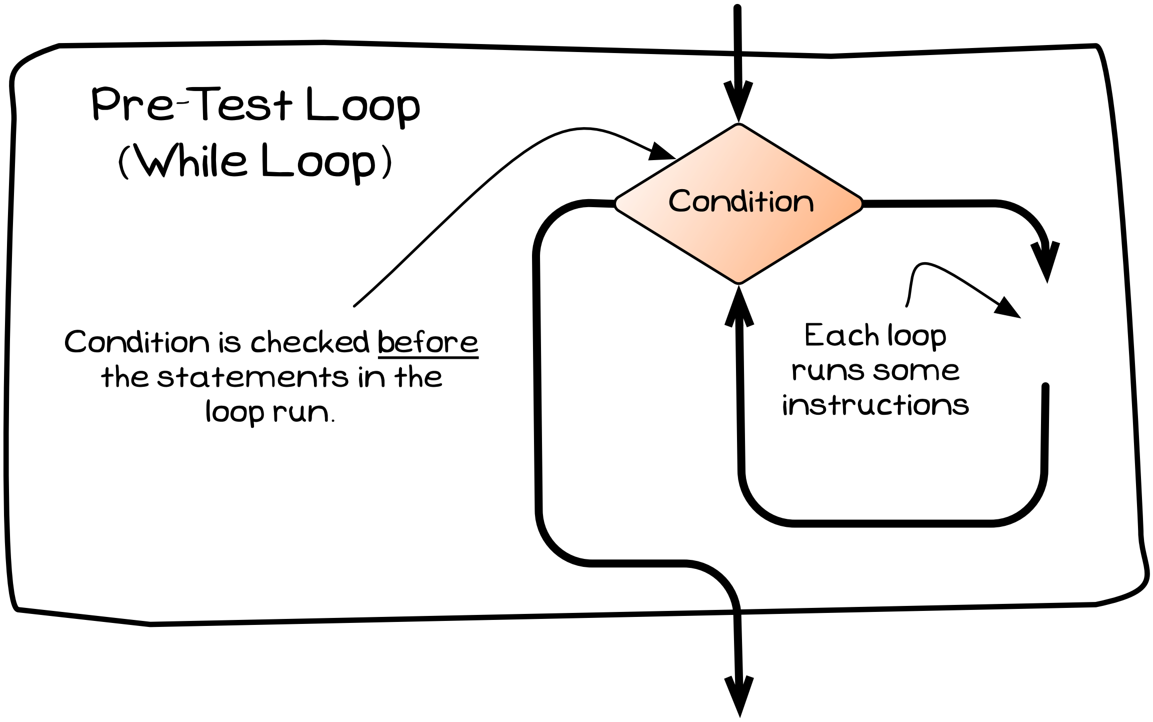

Condition is an Expression that controls that path is taken...
Branching
Condition
Each path can contain other instructions
Paths converge after the branch


Statement
Is a kind of Statement
Condition is a Boolean Expression
If Statement
The False path can optionally have instructions in an else path
Condition
True
False
else
This path is taken when the condition is True
If there is no else part the if statement just skips the True path when the condition is False


Statement
Is a kind of Statement
Expression selects the path with the matching value
Case Statement
Expression
1
2
3
An alternate path exists for when the Expression does not match any path's value
This path is taken when the value of the expression is
You can optionally provide instructions that run when no other path is matched.


Is a kind of Statement
Statement
A Single Statement...
Compound Statement
Statement
Statement
Statement
That contains multiple statements


Each loop
runs some
instructions
Looping
Condition is checked each
loop, and controls if the
loop runs again, or the loop ends
Condition

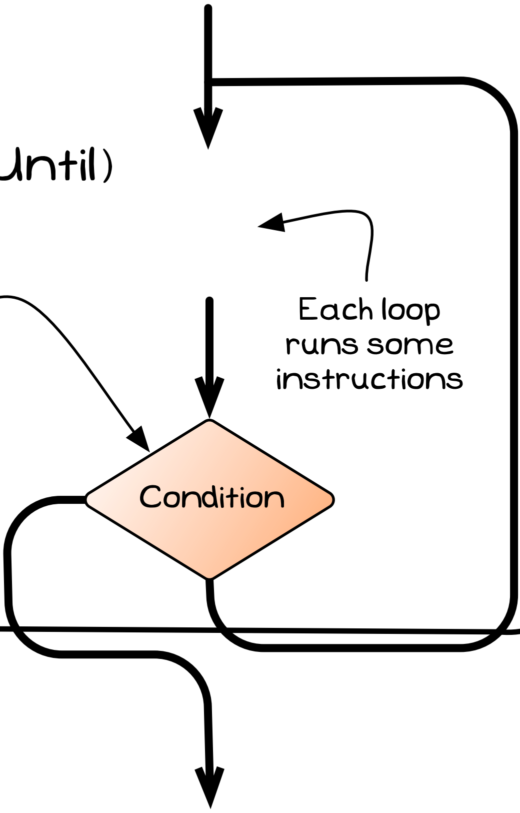


Pre-Test Loop
(While Loop)

Condition is checked before the statements in the loop run.

Each loop runs some instructions

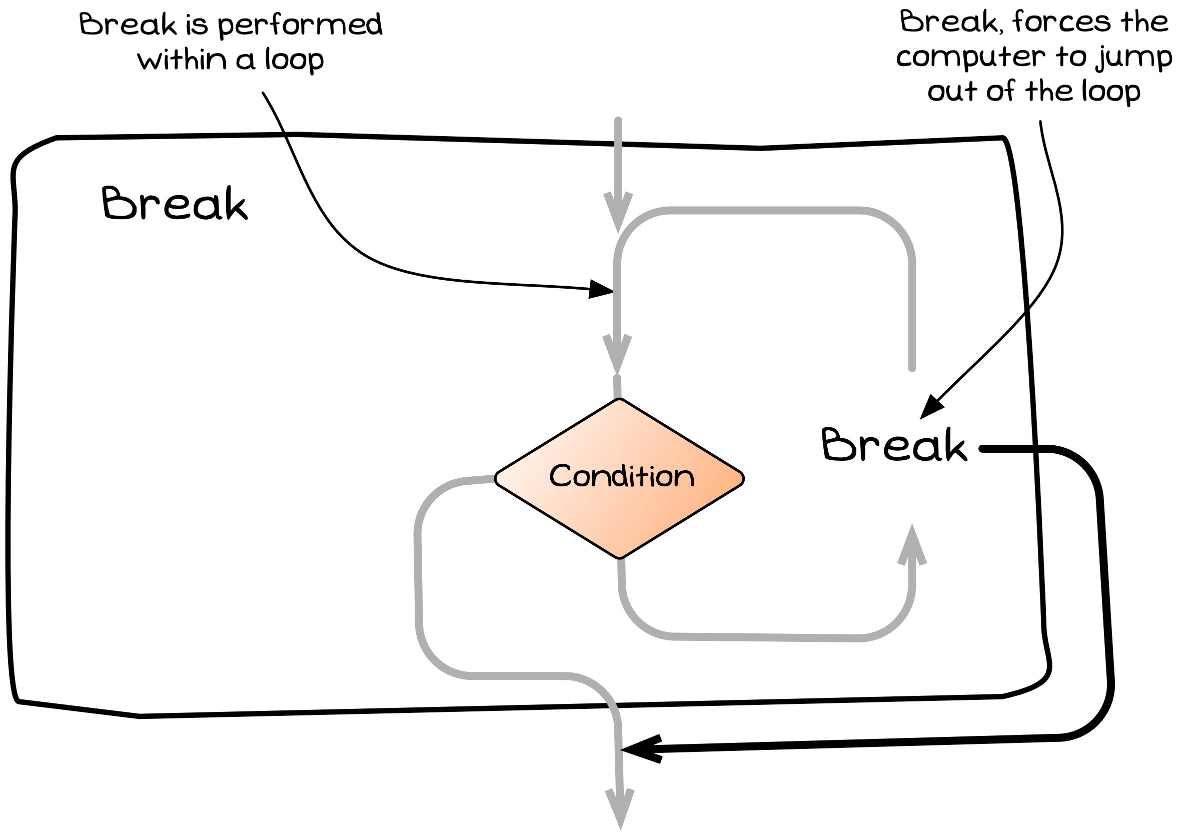
Post-Test Loop
(Do ..While or Repeat... Until)

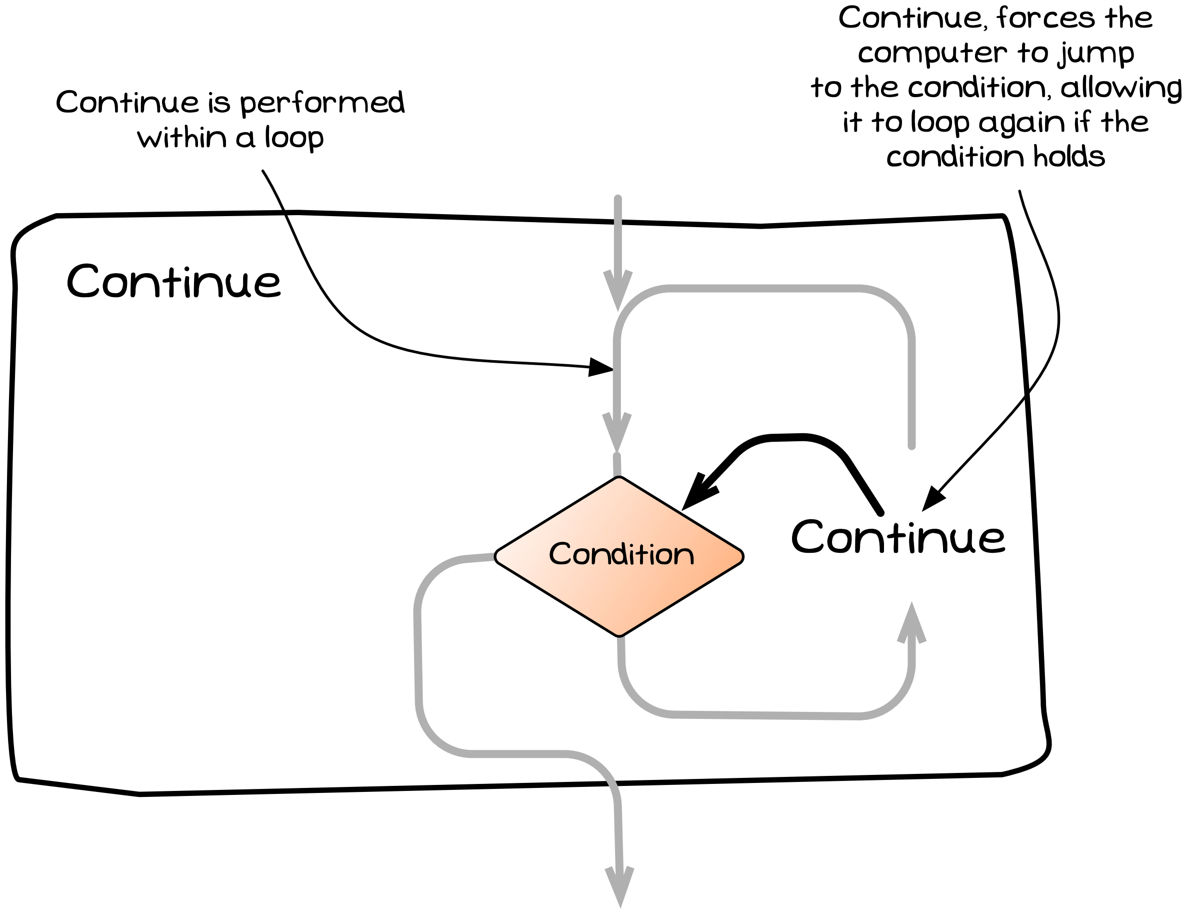


Condition is checked after the statements in the loop run.

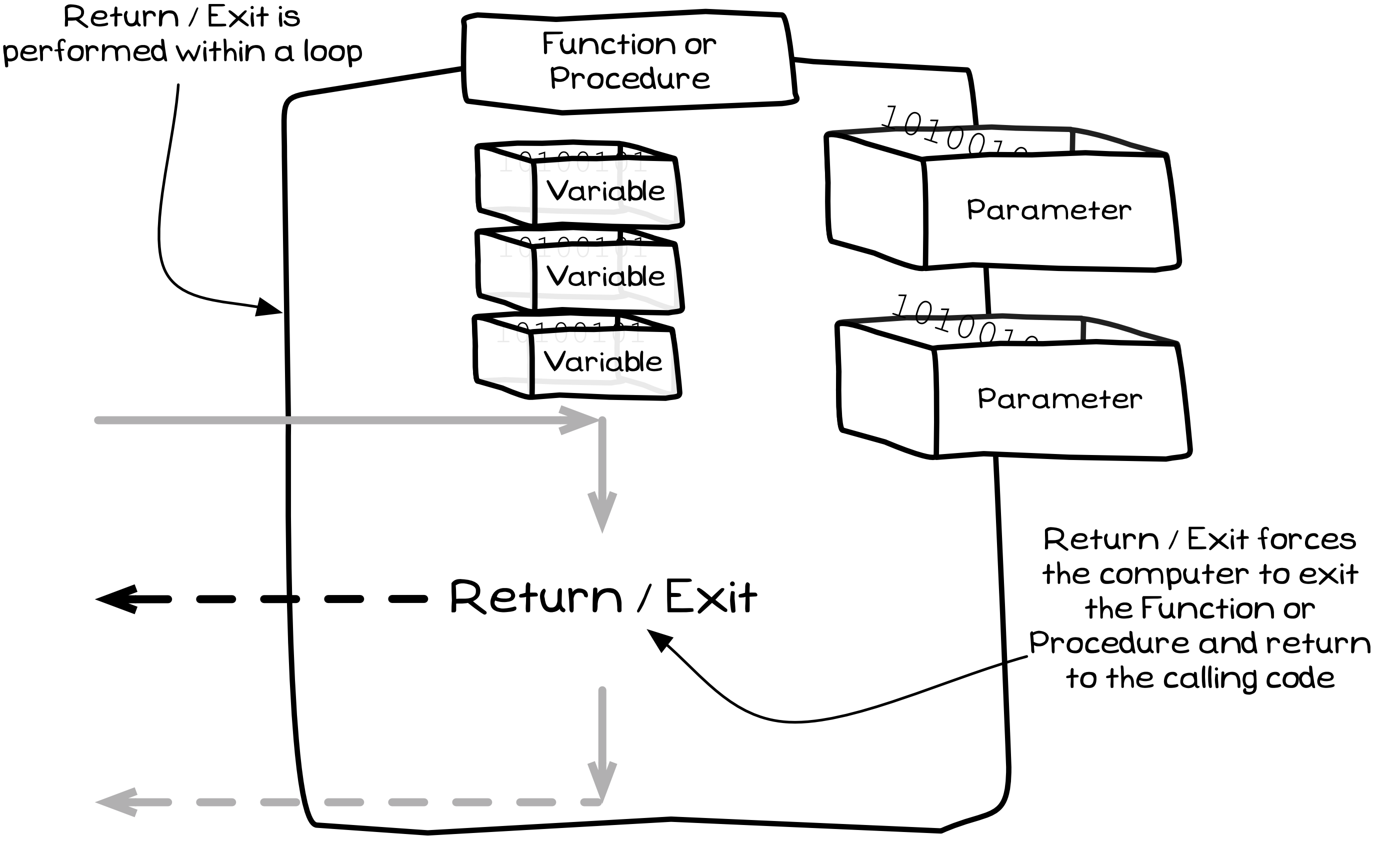
Each loop runs some instructions

Condition


Break is performed within a loop
Break, forces the computer to jump out of the loop
Break
Condition
Break


Continue, forces the computer to jump to the condition, allowing it to loop again if the condition holds
Continue is performed within a loop
Continue
Condition
Continue

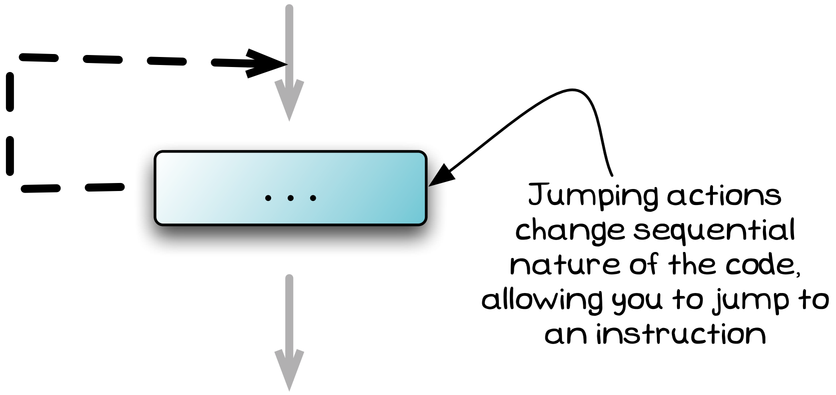
Return / Exit



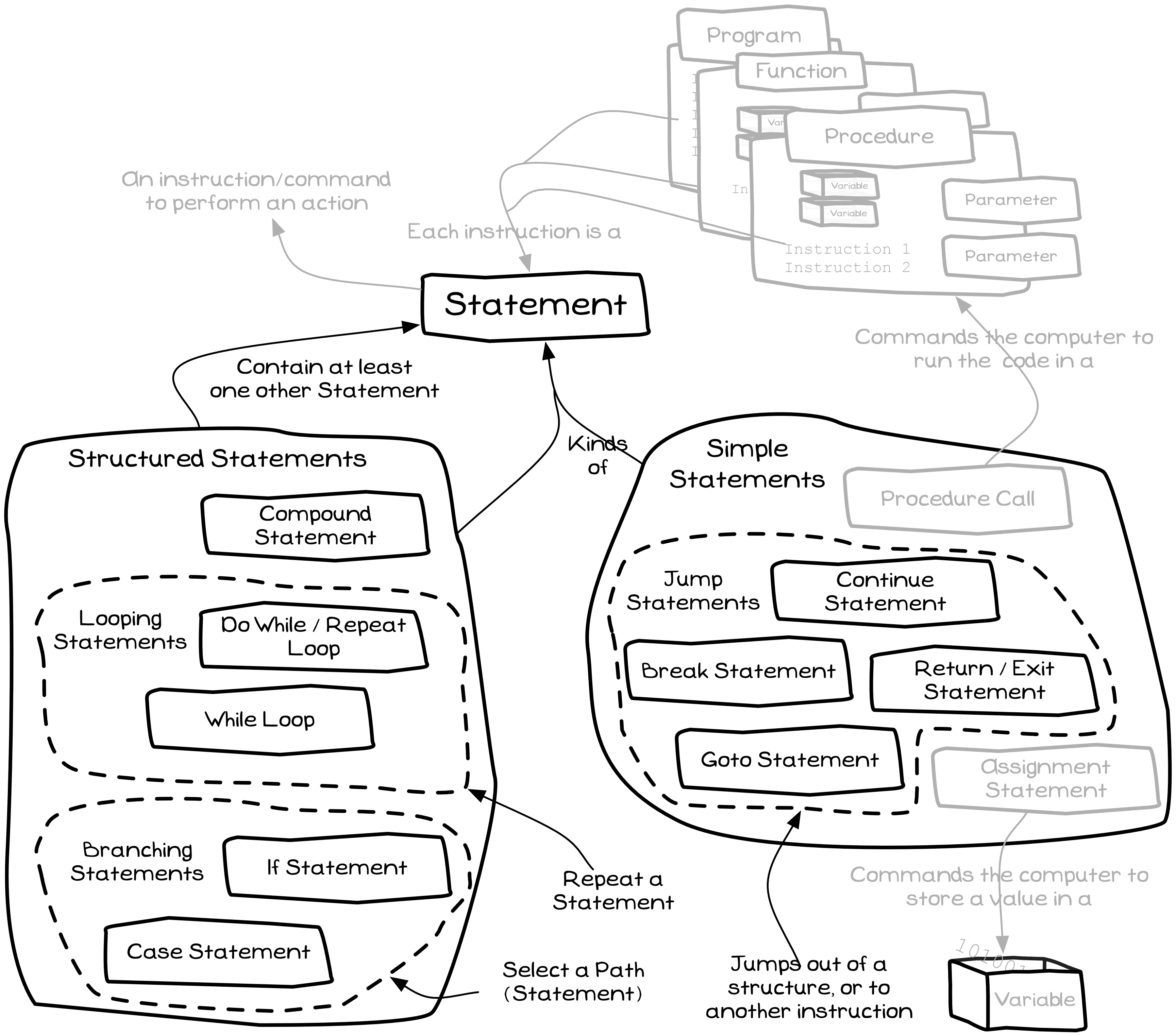
Return / Exit is performed within a loop

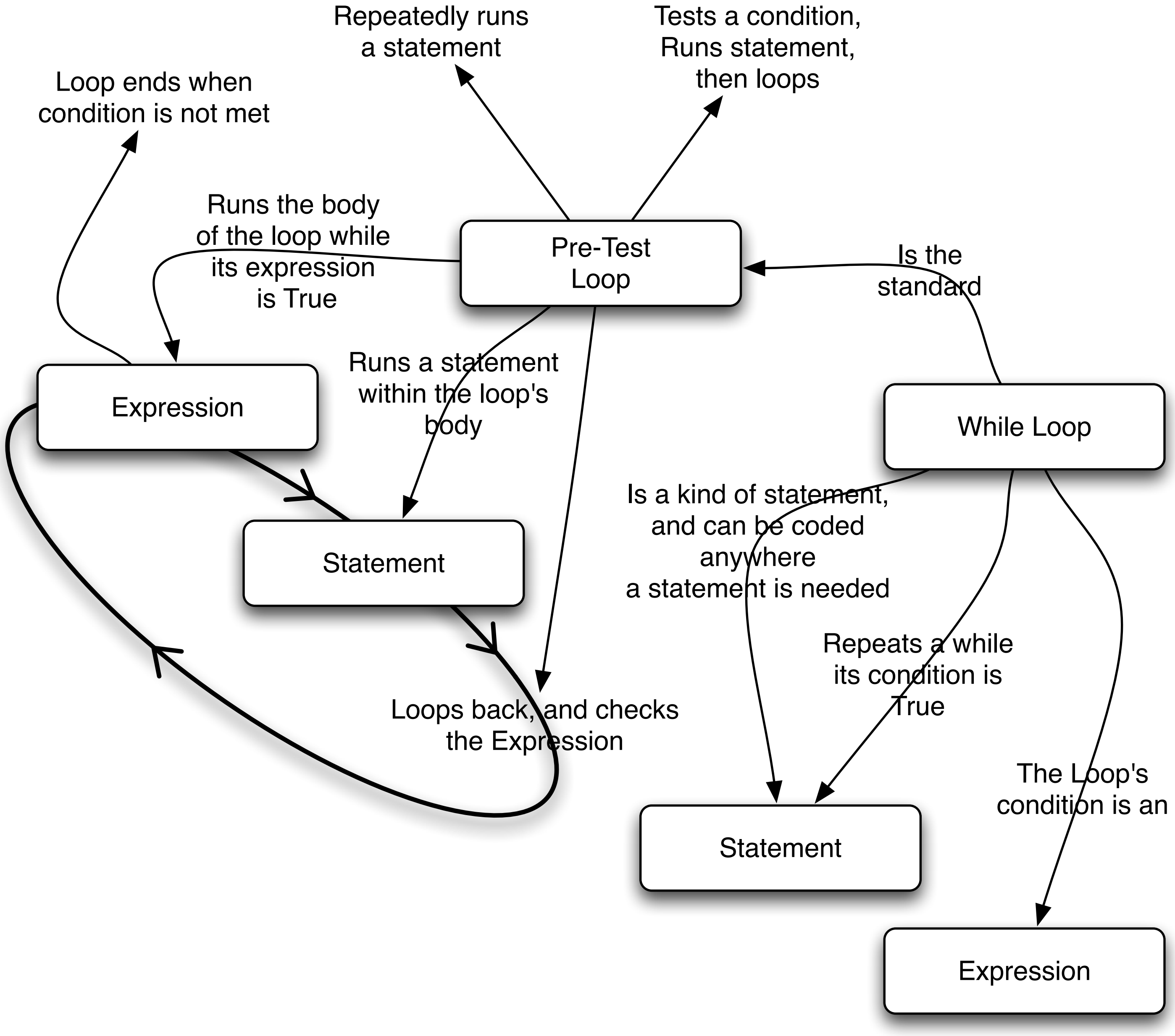
Return / Exit forces the computer to exit the Function or Procedure and return to the calling code

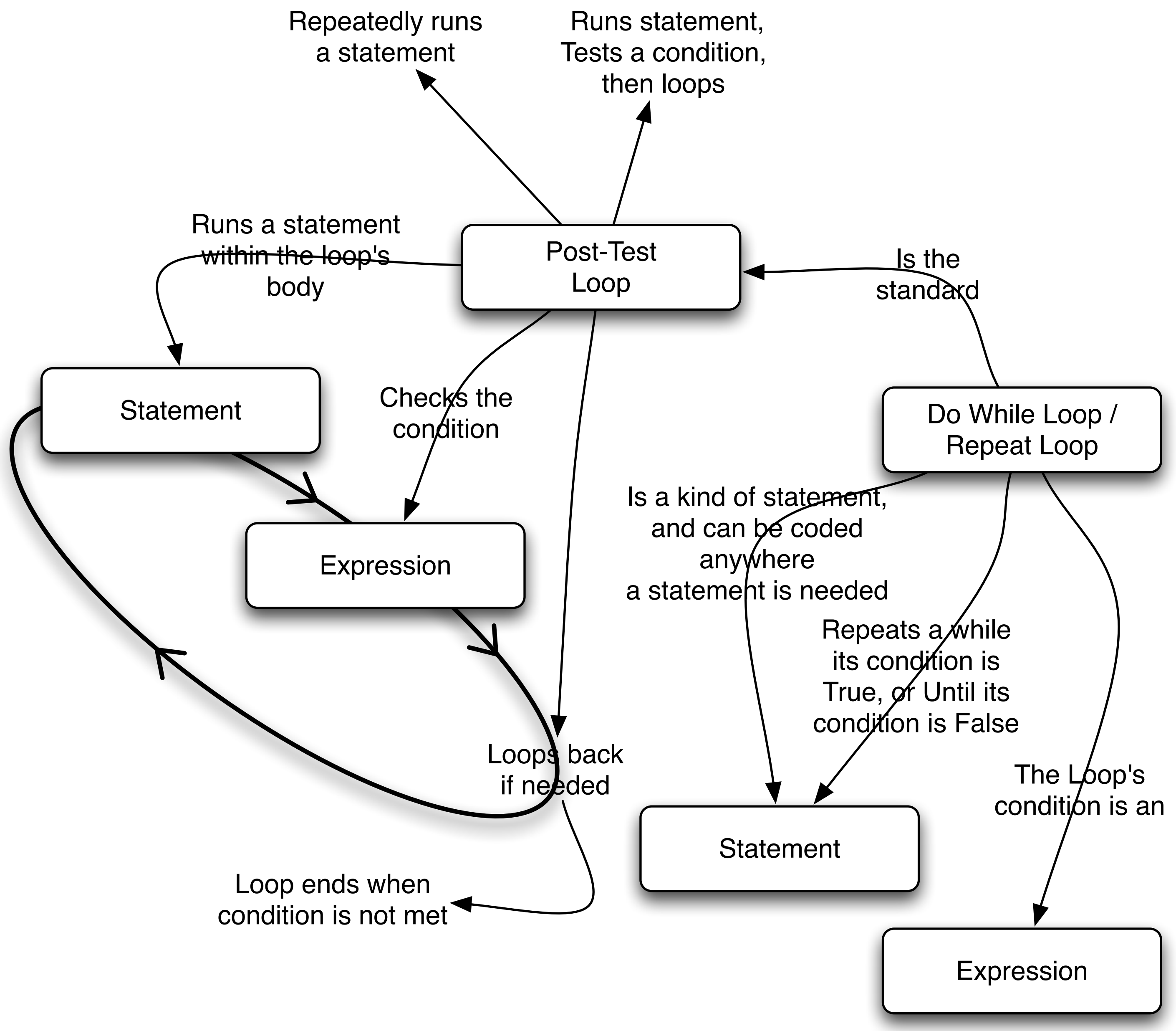
Jumping



Jumping actions change sequential nature of the code, allowing you to jump to an instruction


Program
Function
Procedure
Variable
Variable
Parameter
Parameter
Instruction 1
Instruction 2
An instruction/command to perform an action
Each instruction is a
Statement
Commands the computer to run the code in a
Contain at least one other Statement
Kinds of
Structured Statements
Compound Statement
Looping Statements
Do While / Repeat Loop
While Loop
Branching Statements
If Statement
Case Statement
Repeat a Statement
Select a Path (Statement)
Simple Statements
Procedure Call
Jump Statements
Continue Statement
Break Statement
Return / Exit Statement
Goto Statement
Assignment Statement
Jumps out of a structure, or to another instruction
Commands the computer to store a value in a
1010
Variable


Repeatedly runs a statement
Tests a condition, Runs statement, then loops
Loop ends when condition is not met
Runs the body of the loop while its expression is True
Pre-Test Loop
Is the standard
Expression
Runs a statement within the loop's body
While Loop
Statement
Is a kind of statement, and can be coded anywhere a statement is needed
Repeats a while its condition is True
Loops back, and checks the Expression
The Loop's condition is an
Statement
Expression



Repeatedly runs a statement

Runs statement, Tests a condition, then loops

Post-Test Loop

Runs a statement within the loop's body

Is the standard

Statement

Checks the condition

Do While Loop / Repeat Loop

Expression

Is a kind of statement, and can be coded anywhere a statement is needed

Repeats a while its condition is True, or Until its condition is False

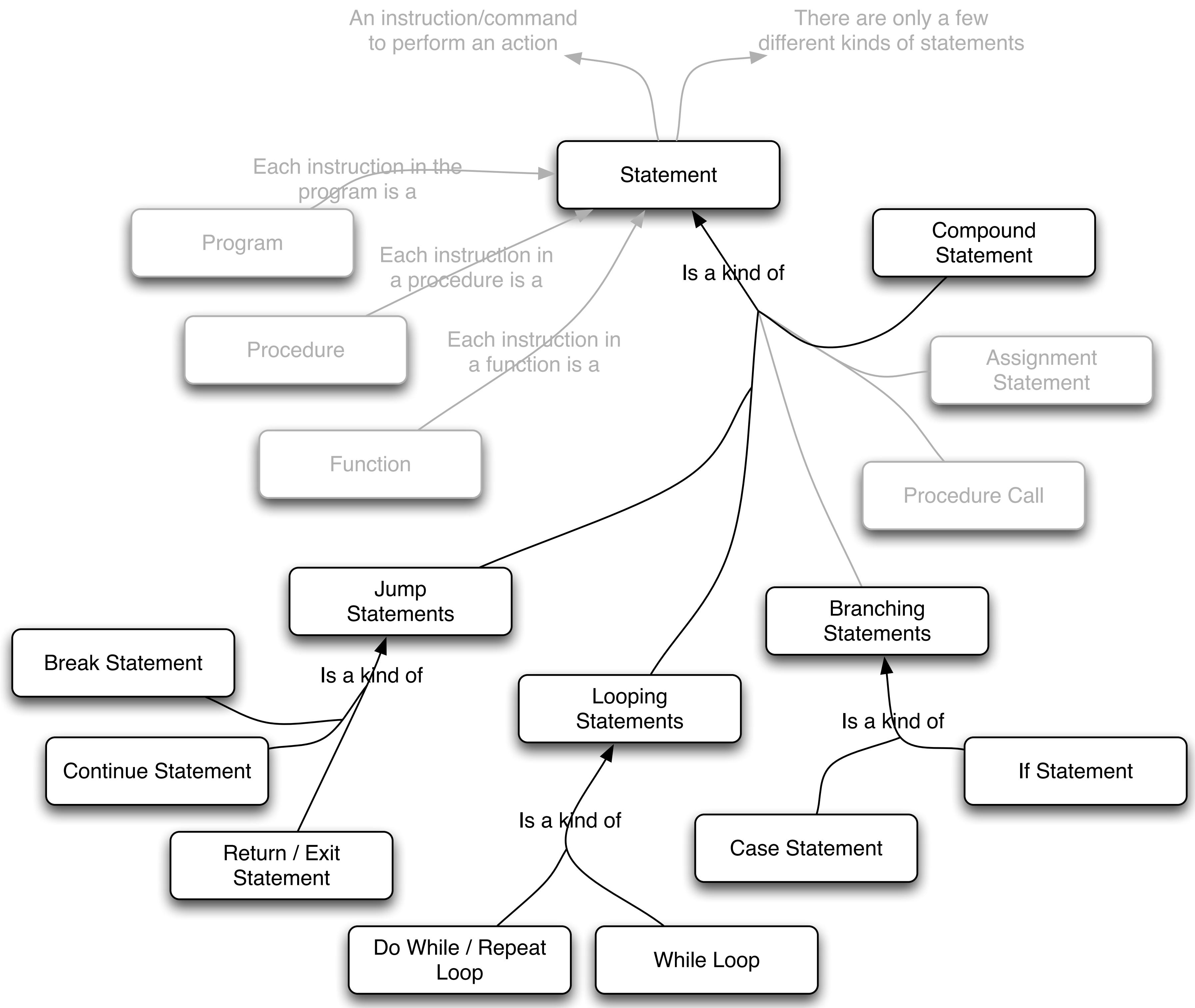
Loops back if needed

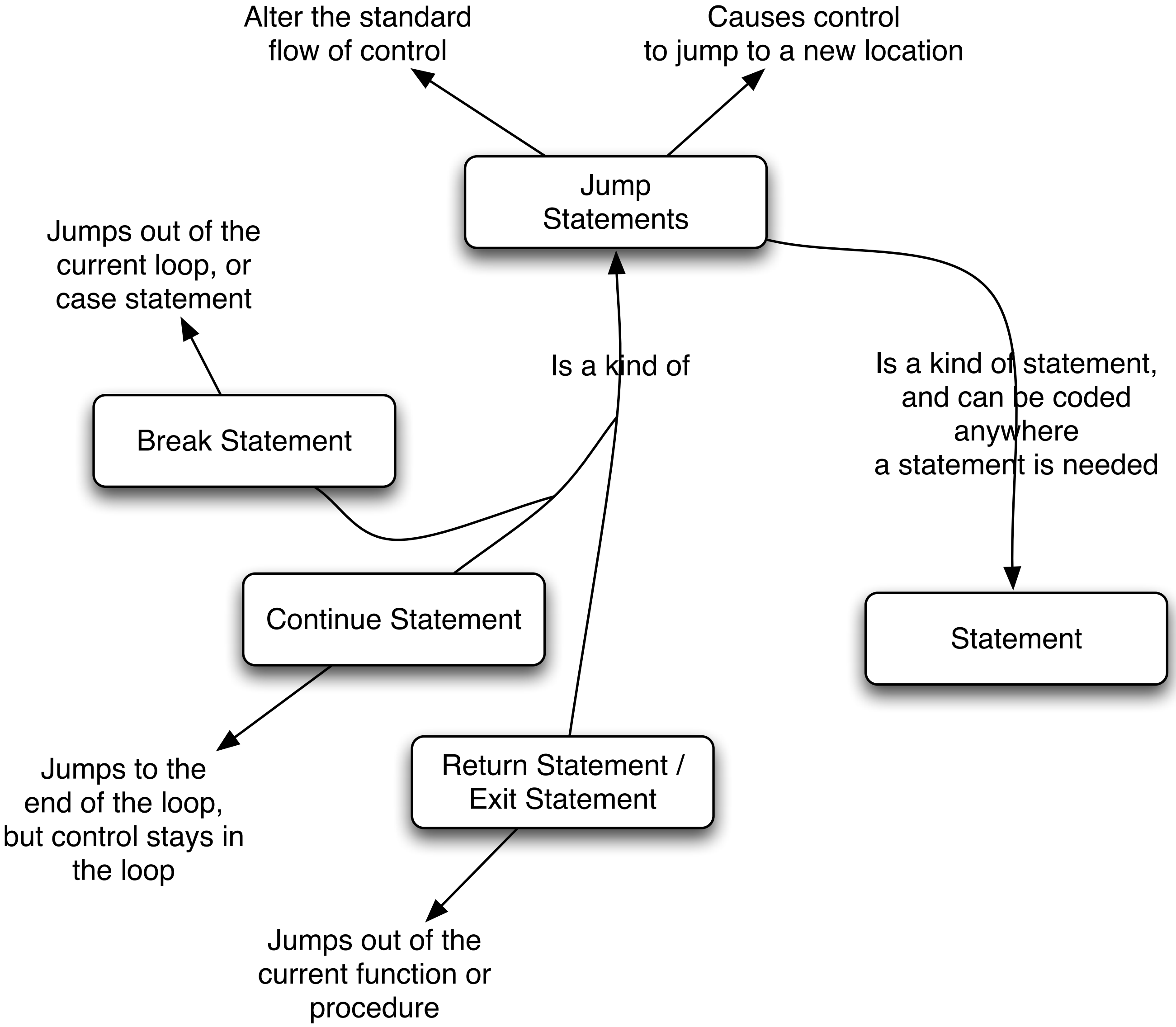
The Loop's condition is an

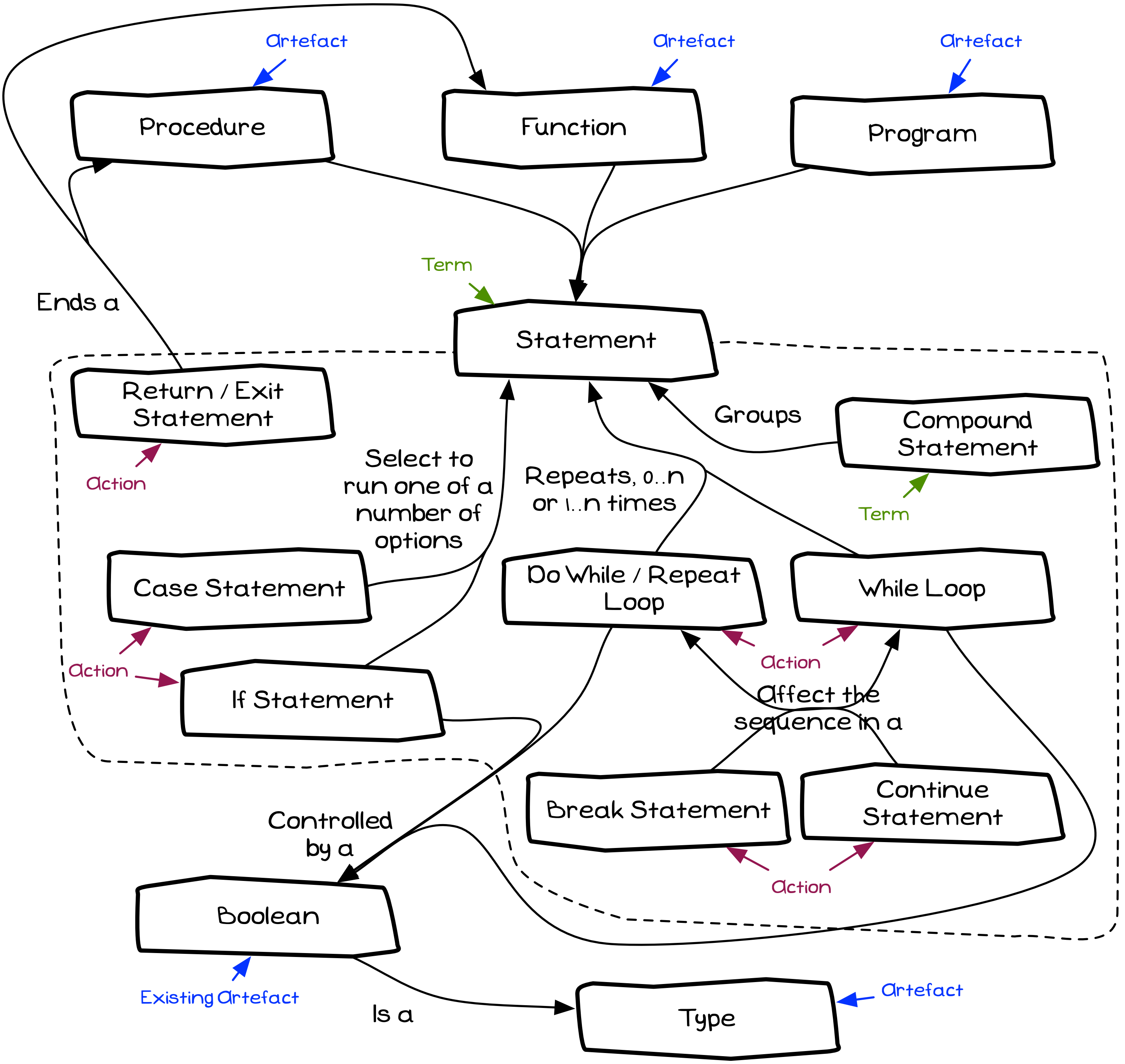
Statement

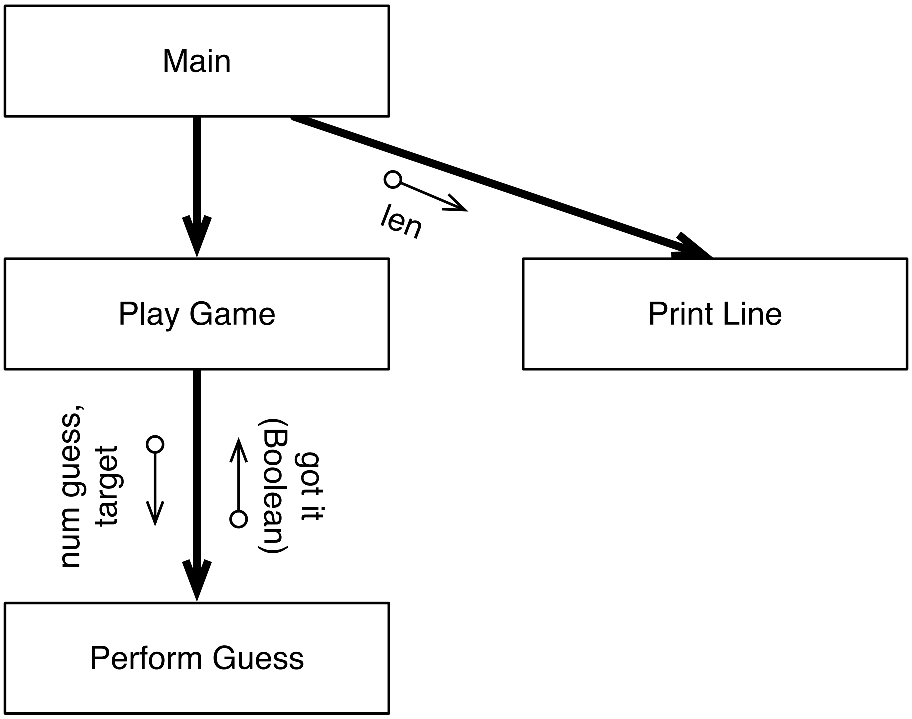
Loop ends when condition is not met

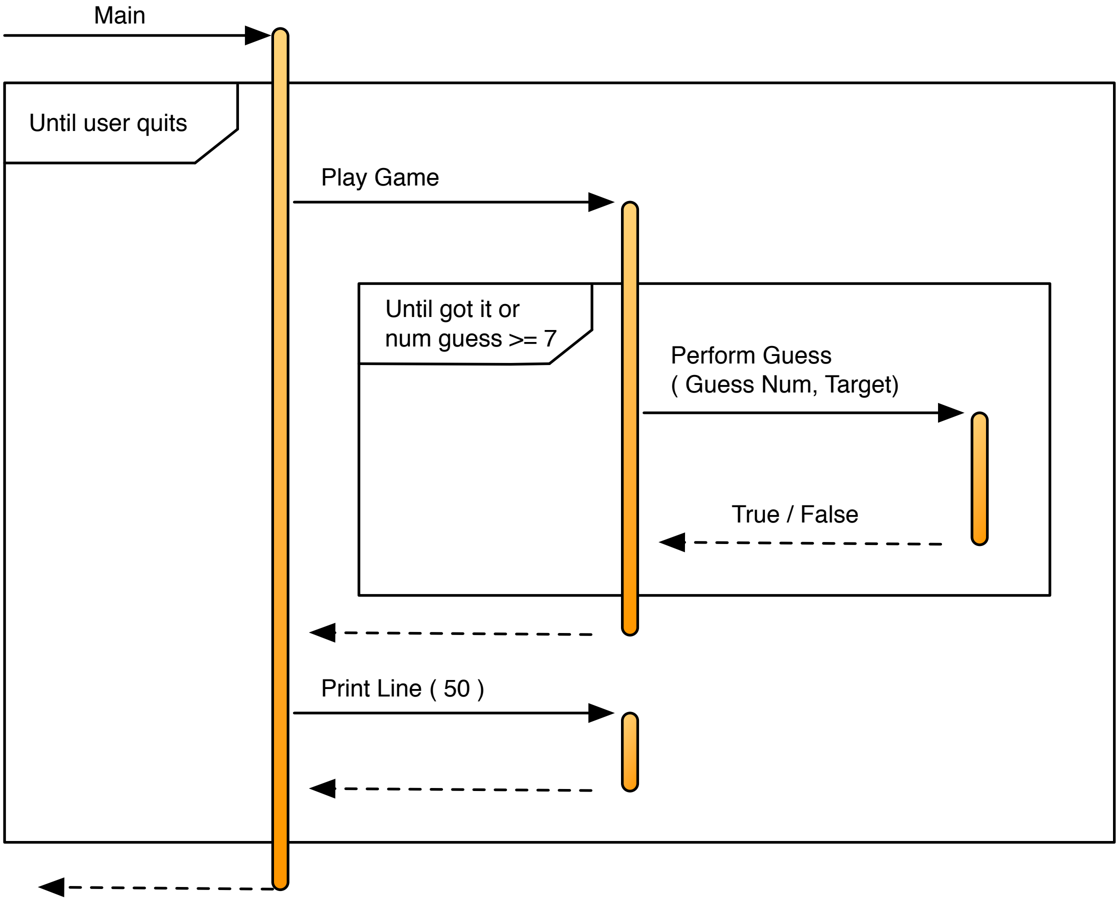
Expression

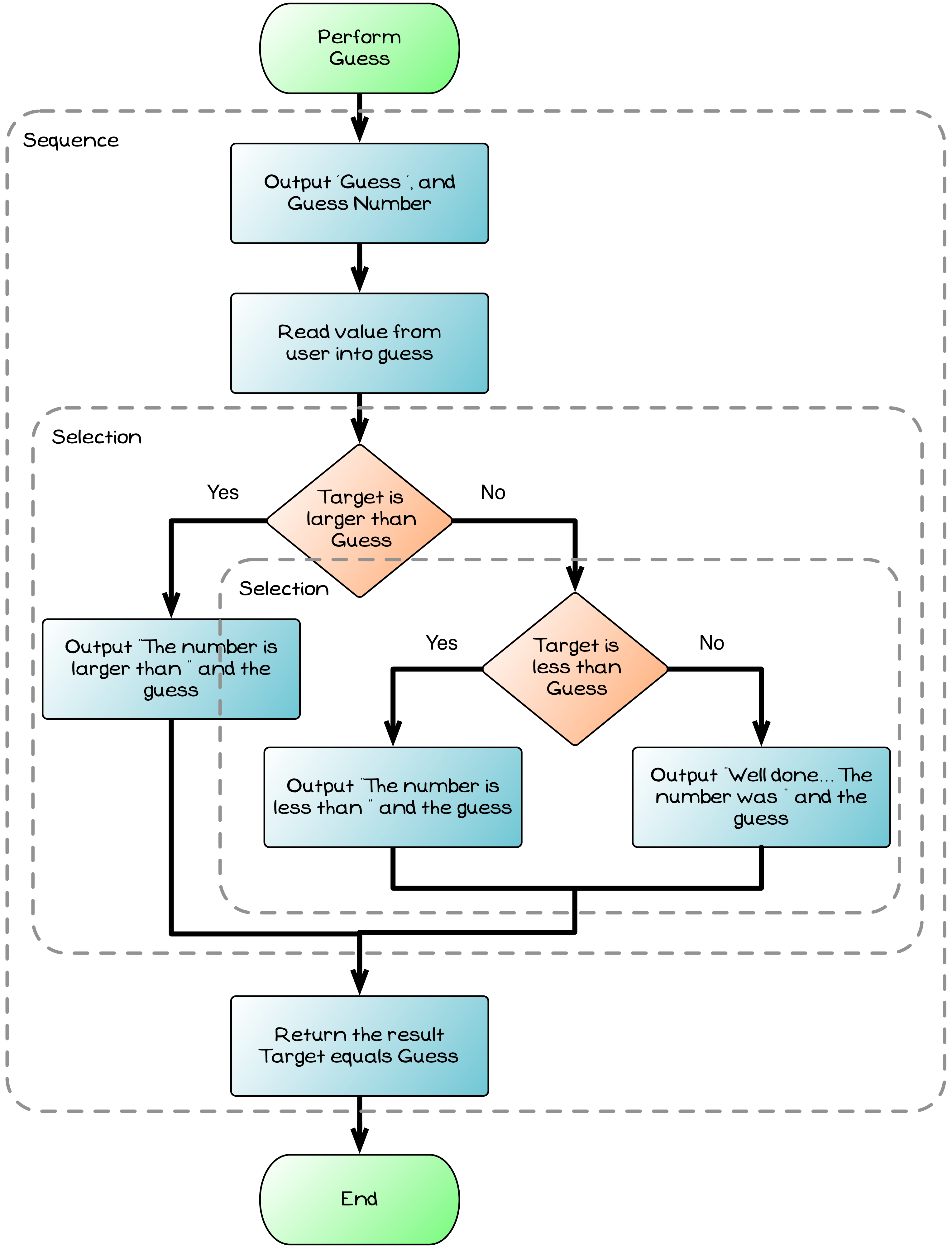

An instruction/command to perform an action
There are only a few different kinds of statements
Statement
Each instruction in the program is a
Program
Each instruction in a procedure is a
Procedure
Each instruction in a function is a
Function
Is a kind of
Compound Statement
Assignment Statement
Procedure Call
Jump Statements
Looping Statements
Branching Statements
Break Statement
Is a kind of
Continue Statement
Return / Exit Statement
Is a kind of
Do While / Repeat Loop
While Loop
Is a kind of
If Statement
Case Statement

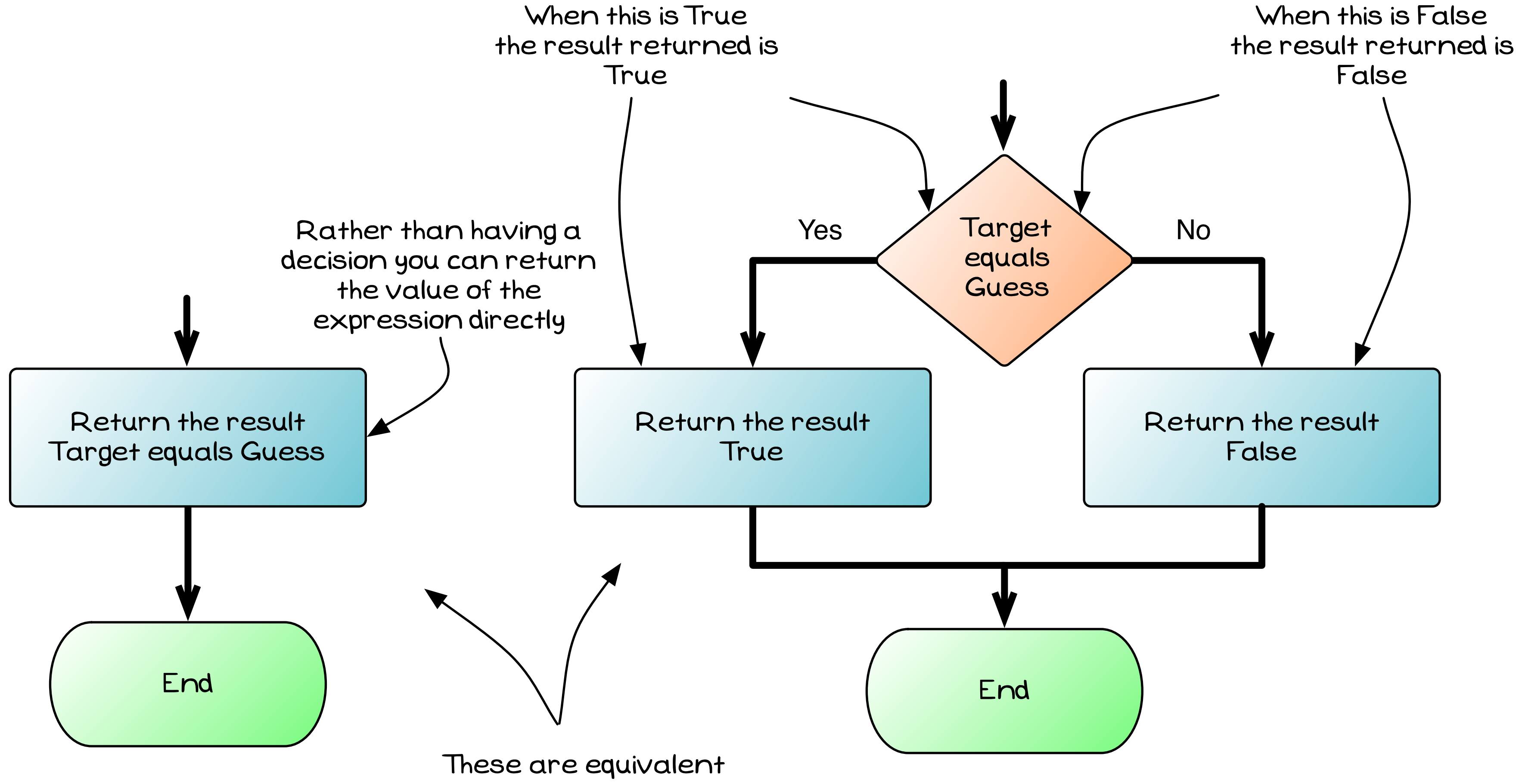

Alter the standard flow of control
Causes control to jump to a new location
Jump Statements
Jumps out of the current loop, or case statement
Is a kind of
Is a kind of statement, and can be coded anywhere a statement is needed
Break Statement
Continue Statement
Statement
Jumps to the end of the loop, but control stays in the loop
Return Statement / Exit Statement
Jumps out of the current function or procedure

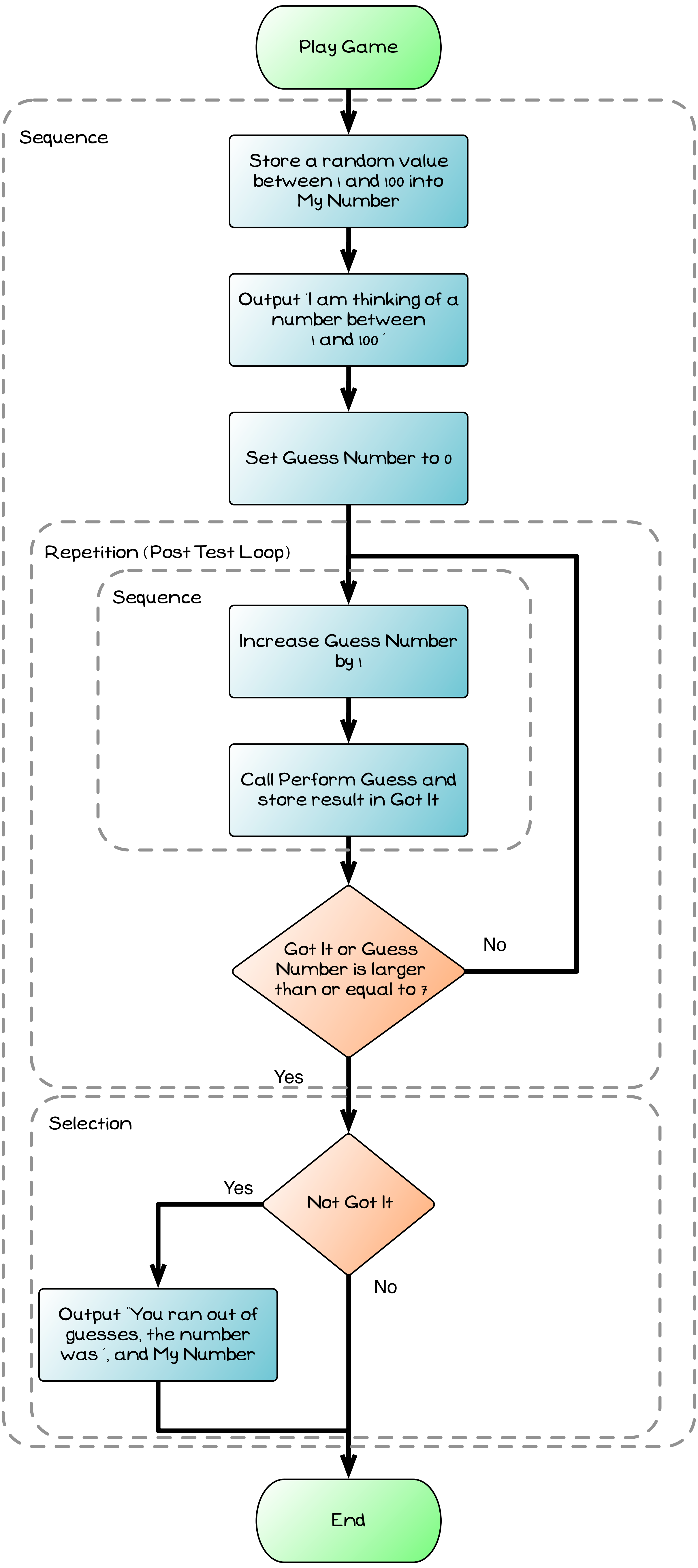

Artefact
Artefact
Artefact
Procedure
Function
Program
Term
Ends a
Statement
Return / Exit Statement
Groups
Compound Statement
Action
Select to run one of a number of options
Repeats, 0..n or 1..n times
Term
Case Statement
Do While / Repeat Loop
While Loop
Action
Action
Action
If Statement
Affect the sequence in a
Break Statement
Continue Statement
Controlled by a
Action
Boolean
Existing Artefact
Is a
Type
Artefact

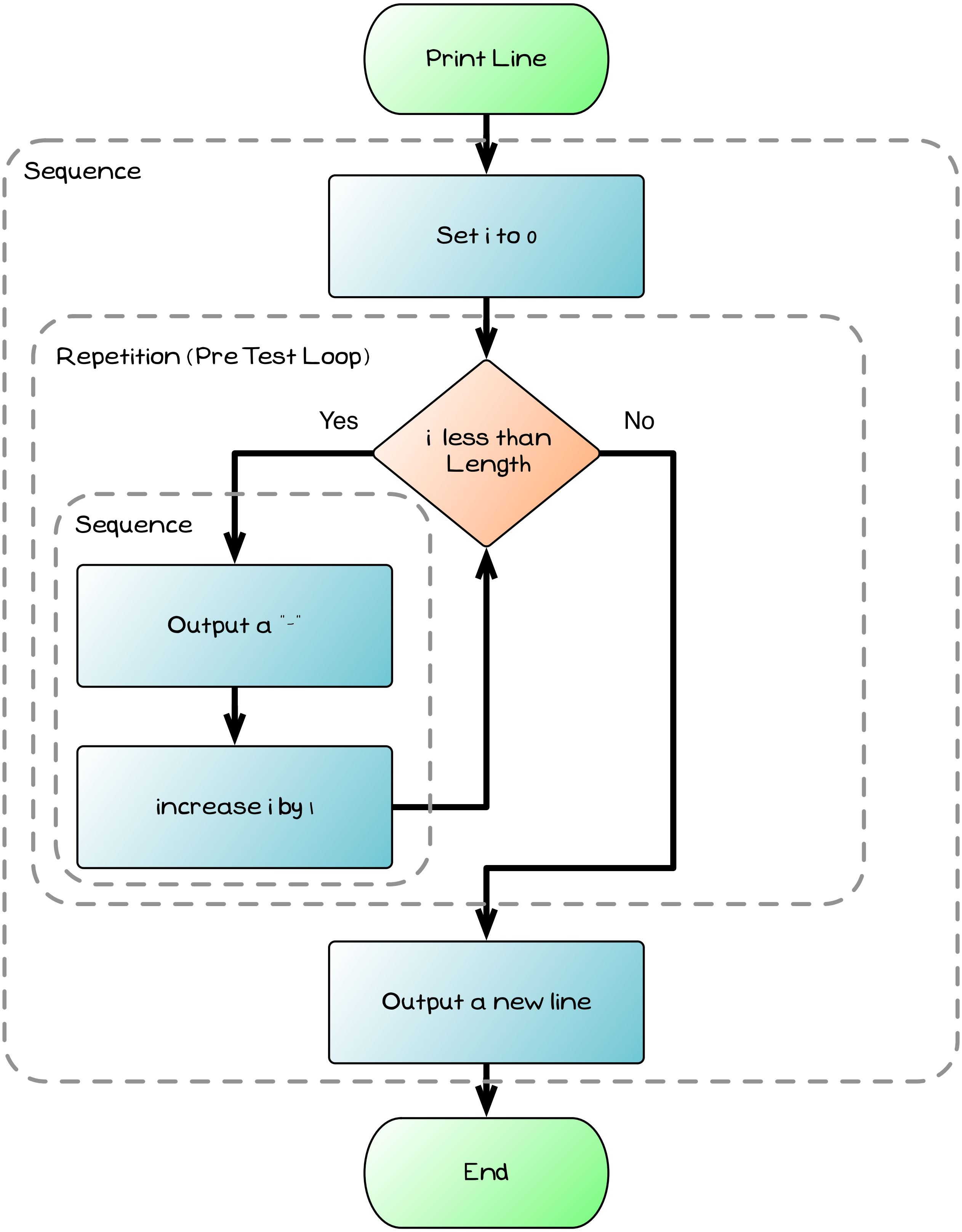

Main
Play Game
Print Line
Perform Guess
len
num guess, target
got it (Boolean)

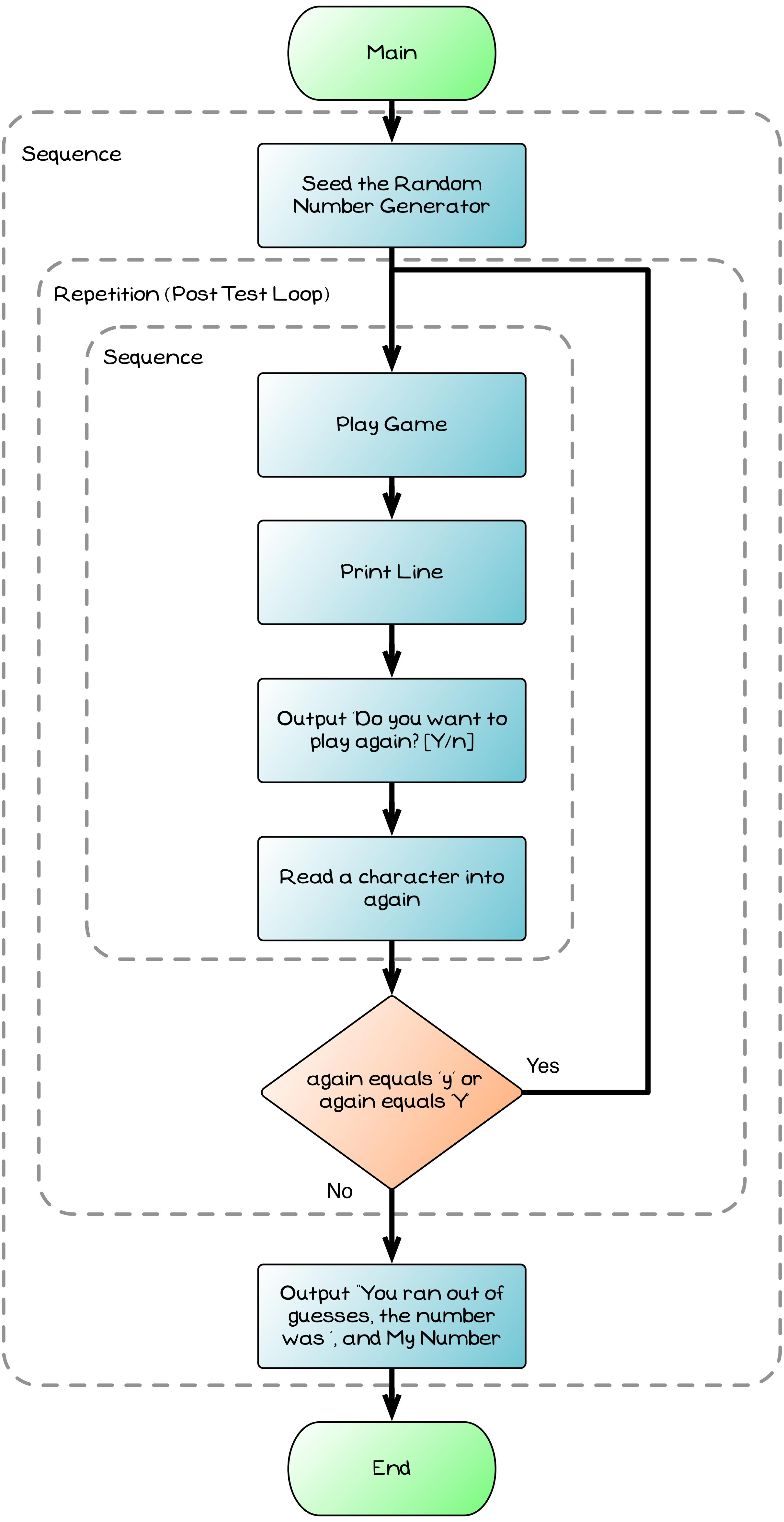

Main
Until user quits
Play Game
Until got it or
num guess >= 7
Perform Guess
(Guess Num, Target)
True / False
Print Line (50)

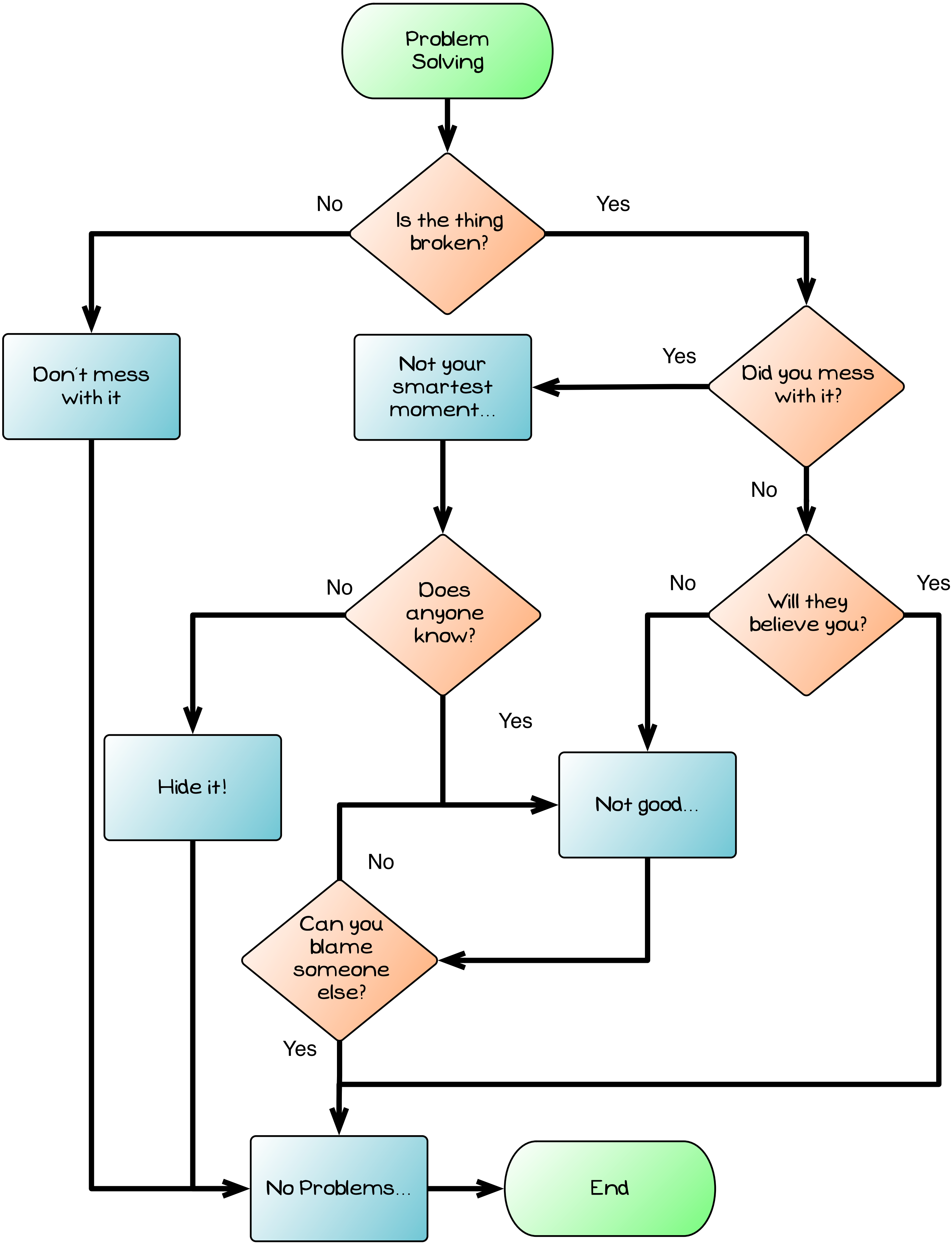

Perform Guess
Sequence
Output 'Guess ', and Guess Number
Read value from user into guess
Selection
Yes
Target is larger than Guess
No
Selection
Output "The number is larger than " and the guess
Yes
Target is less than Guess
No
Output "The number is less than " and the guess
Output "Well done... The number was " and the guess
Return the result Target equals Guess
End

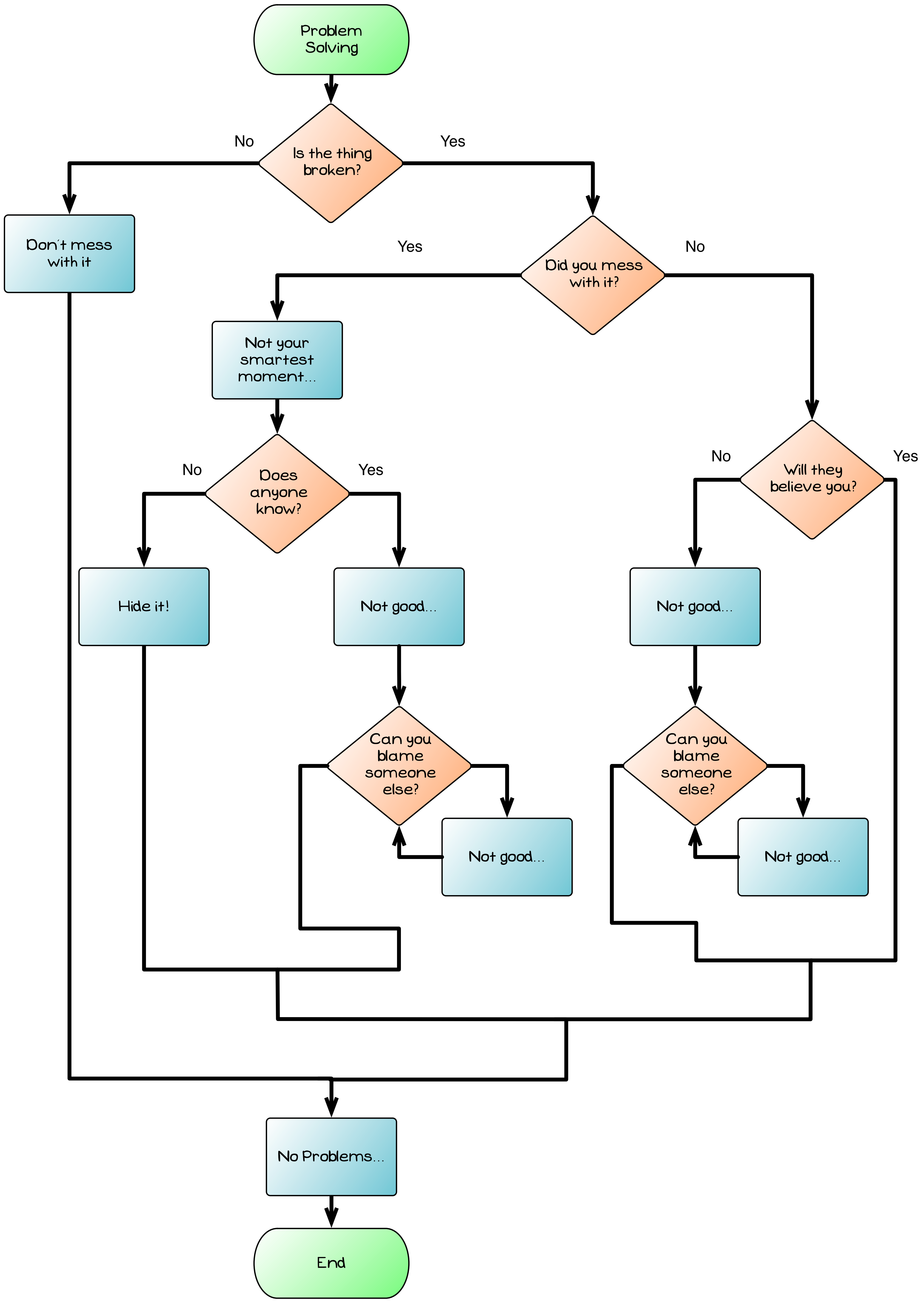

When this is True the result returned is True
When this is False the result returned is False
Rather than having a decision you can return the value of the expression directly
Yes
Target equals Guess
No
Return the result Target equals Guess
Return the result True
Return the result False
End
End
These are equivalent

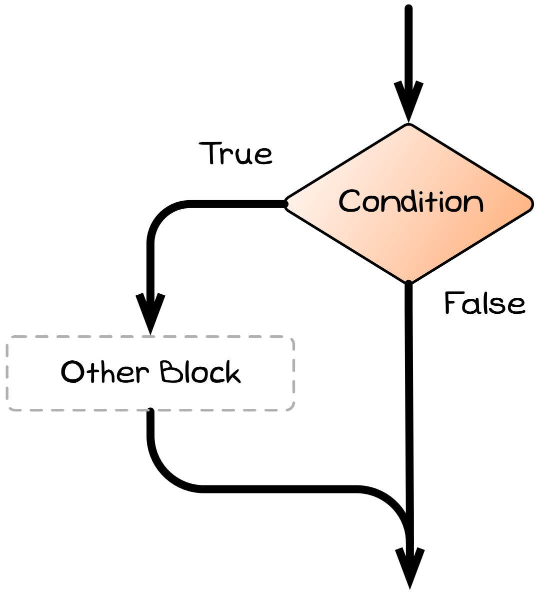

Play Game
Sequence
Store a random value between 1 and 100 into My Number
Output 'I am thinking of a number between 1 and 100'
Set Guess Number to 0
Repetition (Post Test Loop)
Sequence
Increase Guess Number by 1
Call Perform Guess and store result in Got It
Got It or Guess Number is larger than or equal to 7
No
Yes
Selection
Not Got It
Yes
No
Output "You ran out of guesses, the number was ', and My Number
End

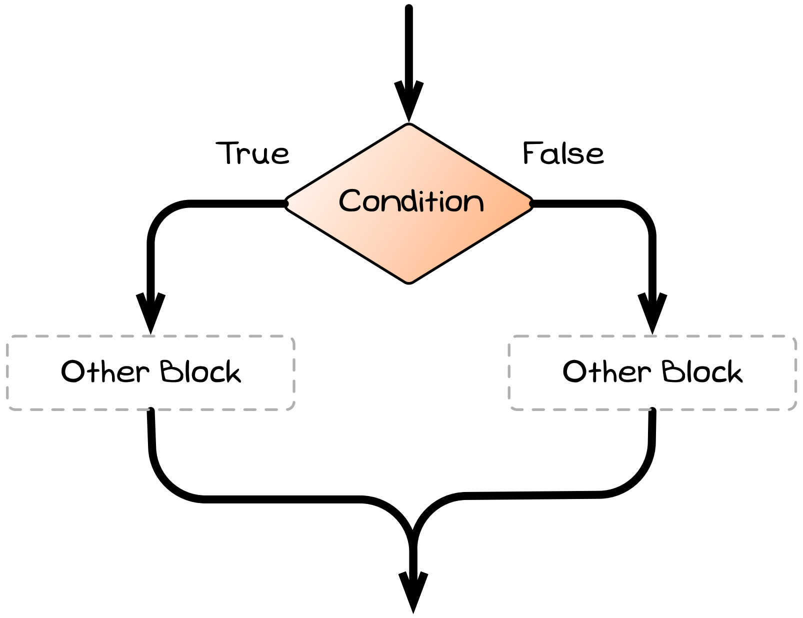

Print Line
Sequence
Set i to 0
Repetition (Pre Test Loop)
Yes
i less than Length
No
Sequence
Output a "-"
increase i by 1
Output a new line
End

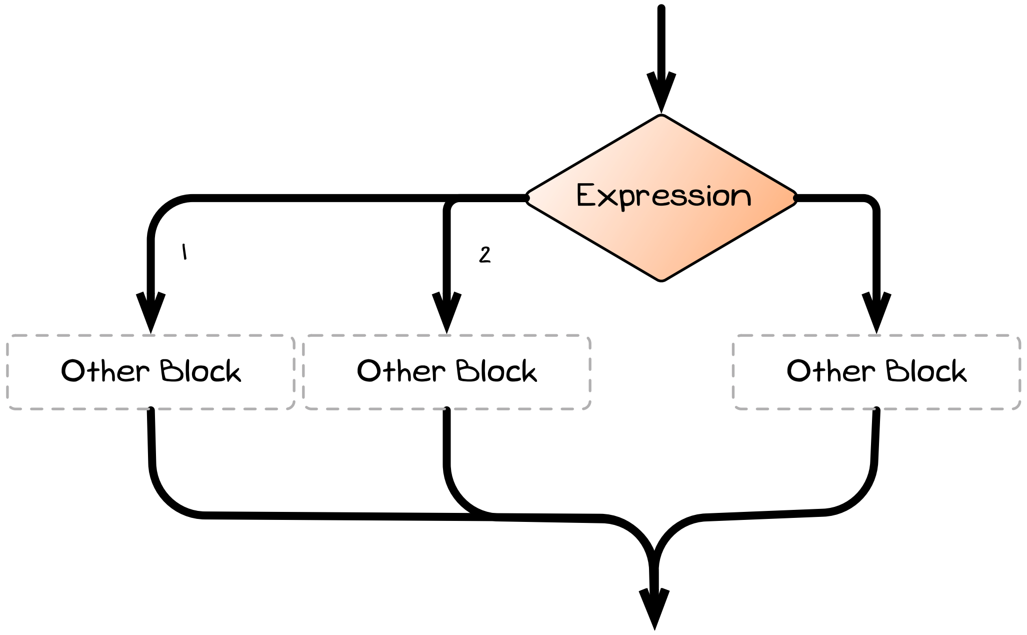

Main
Sequence
Seed the Random Number Generator
Repetition (Post Test Loop)
Sequence
Play Game
Print Line
Output ´Do you want to play again? [Y/n]
Read a character into again
again equals ´y´ or again equals ´Y´
Yes
No
Output `You ran out of guesses, the number was ´, and My Number
End

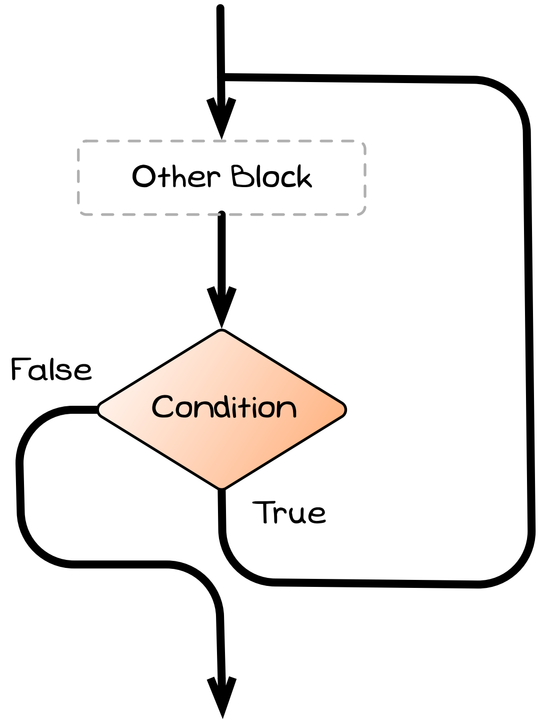

Problem Solving
Is the thing broken?
No
Yes
Don't mess with it
Not your smartest moment...
Yes
Did you mess with it?
No
Does anyone know?
No
No
Will they believe you?
Yes
Yes
Hide it!
Not good...
No
Can you blame someone else?
Yes
No Problems...
End

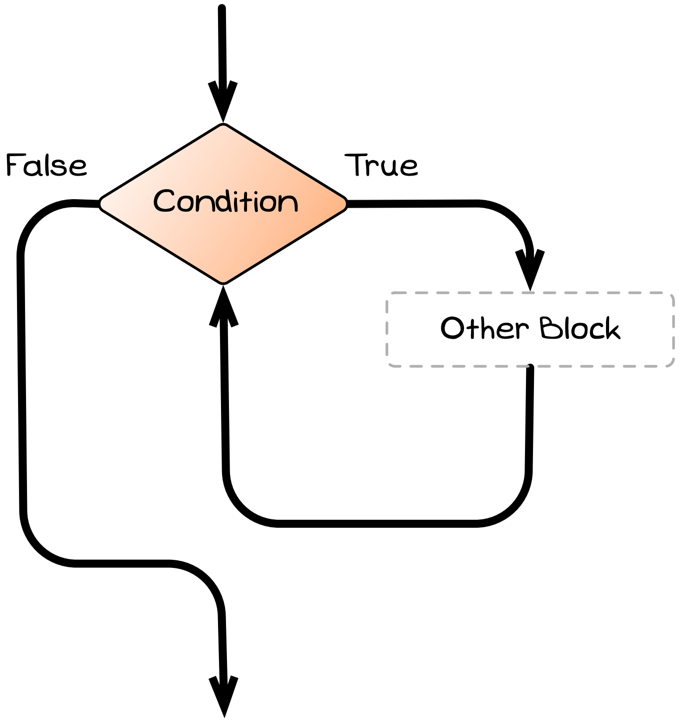

Problem Solving
Is the thing broken?
No
Yes
Don't mess with it
Did you mess with it?
Yes
No
Not your smartest moment...
Does anyone know?
No
Yes
Hide it!
Not good...
Will they believe you?
No
Yes
Not good...
Can you blame someone else?
Not good...
Can you blame someone else?
Not good...
No Problems...
End

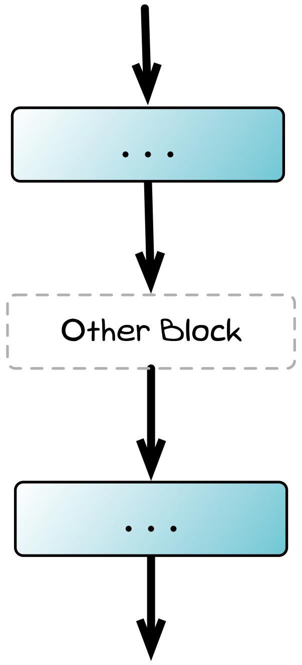

True
Condition
False
Other Block

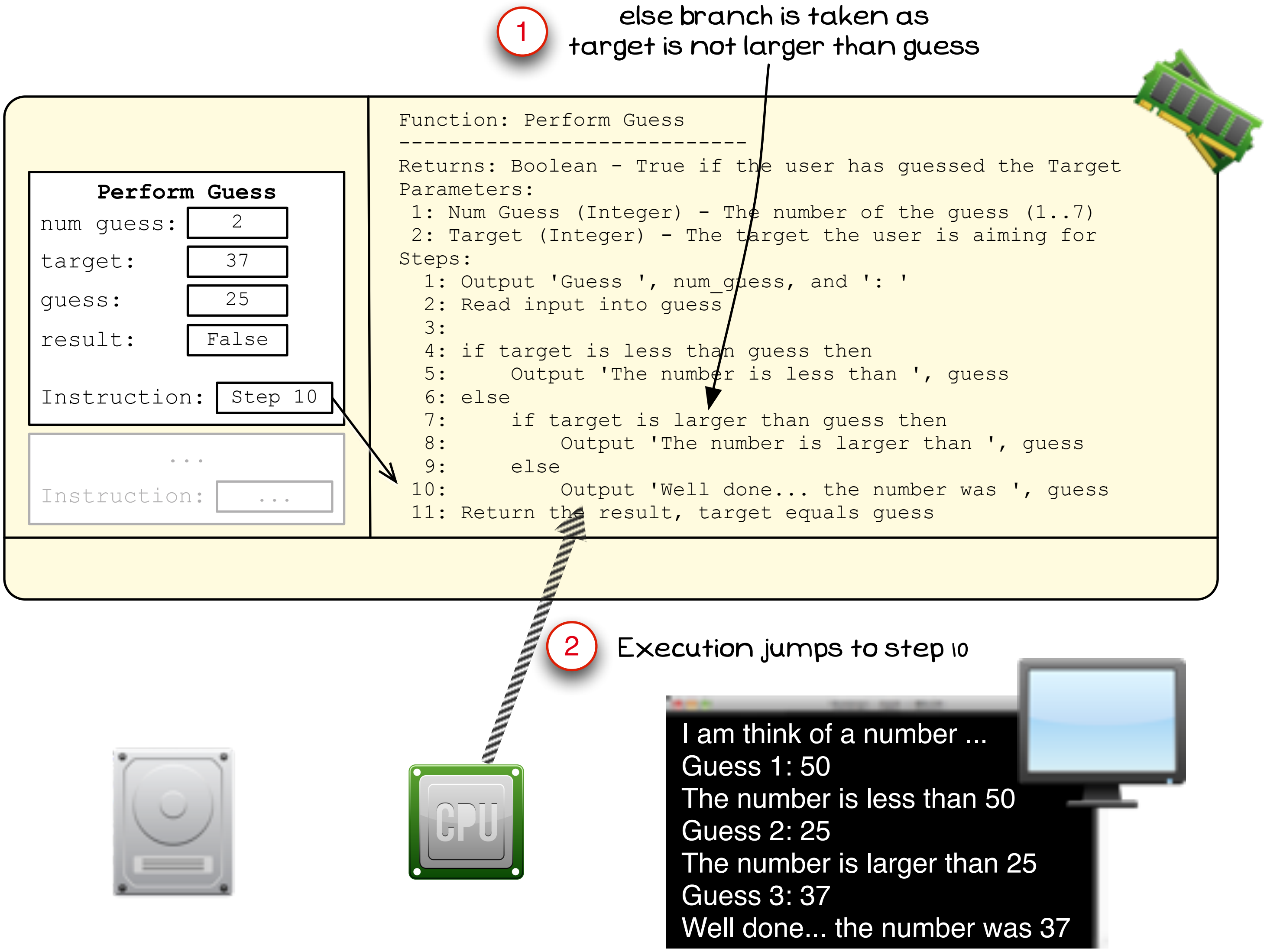

True
Condition
False
Other Block
Other Block


Expression
1
2
Other Block
Other Block
Other Block


Other Block
False
Condition
True


False
Condition
True
Other Block


. . .
Other Block
. . .



1 else branch is taken as target is not larger than guess

Perform Guess

num guess: 2

target: 37

guess: 25

result: False

Instruction: Step 10

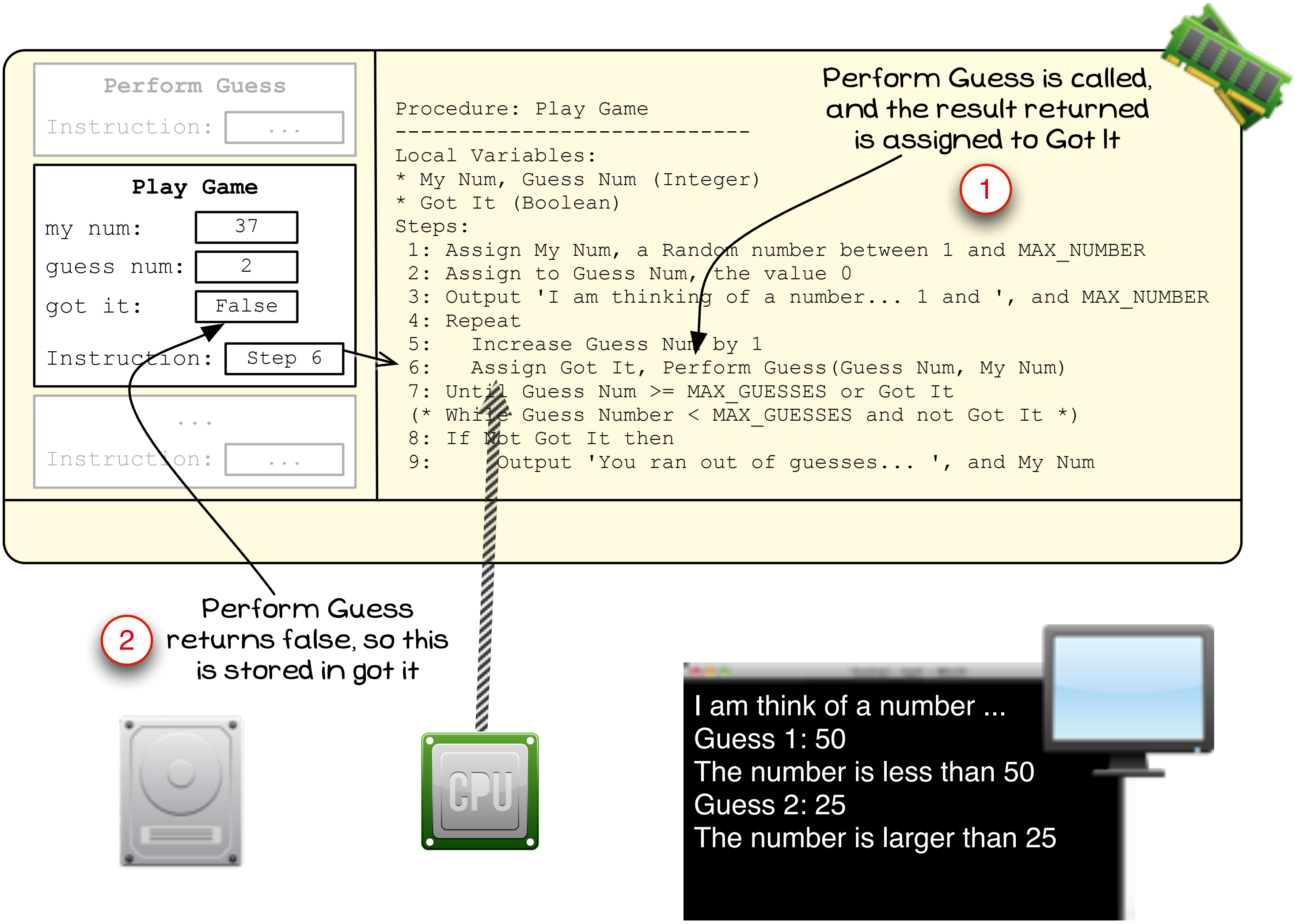
...

Instruction: ...

```
Function: Perform Guess
----------------------------
Returns: Boolean - True if the user has guessed the Target
Parameters:
 1: Num Guess (Integer) - The number of the guess (1..7)
 2: Target (Integer) - The target the user is aiming for
Steps:
  1: Output 'Guess ', num_guess, and ': '
  2: Read input into guess
  3:
  4: if target is less than guess then
  5:     Output 'The number is less than ', guess
  6: else
  7:     if target is larger than guess then
  8:         Output 'The number is larger than ', guess
  9:     else
 10:         Output 'Well done... the number was ', guess
 11: Return the result, target equals guess
```

2 Execution jumps to step 10

```
I am think of a number ...
Guess 1: 50
The number is less than 50
Guess 2: 25
The number is larger than 25
Guess 3: 37
Well done... the number was 37
```



Perform Guess

Instruction: ...

Play Game

my num: 37

guess num: 2

got it: False

Instruction: Step 6

...

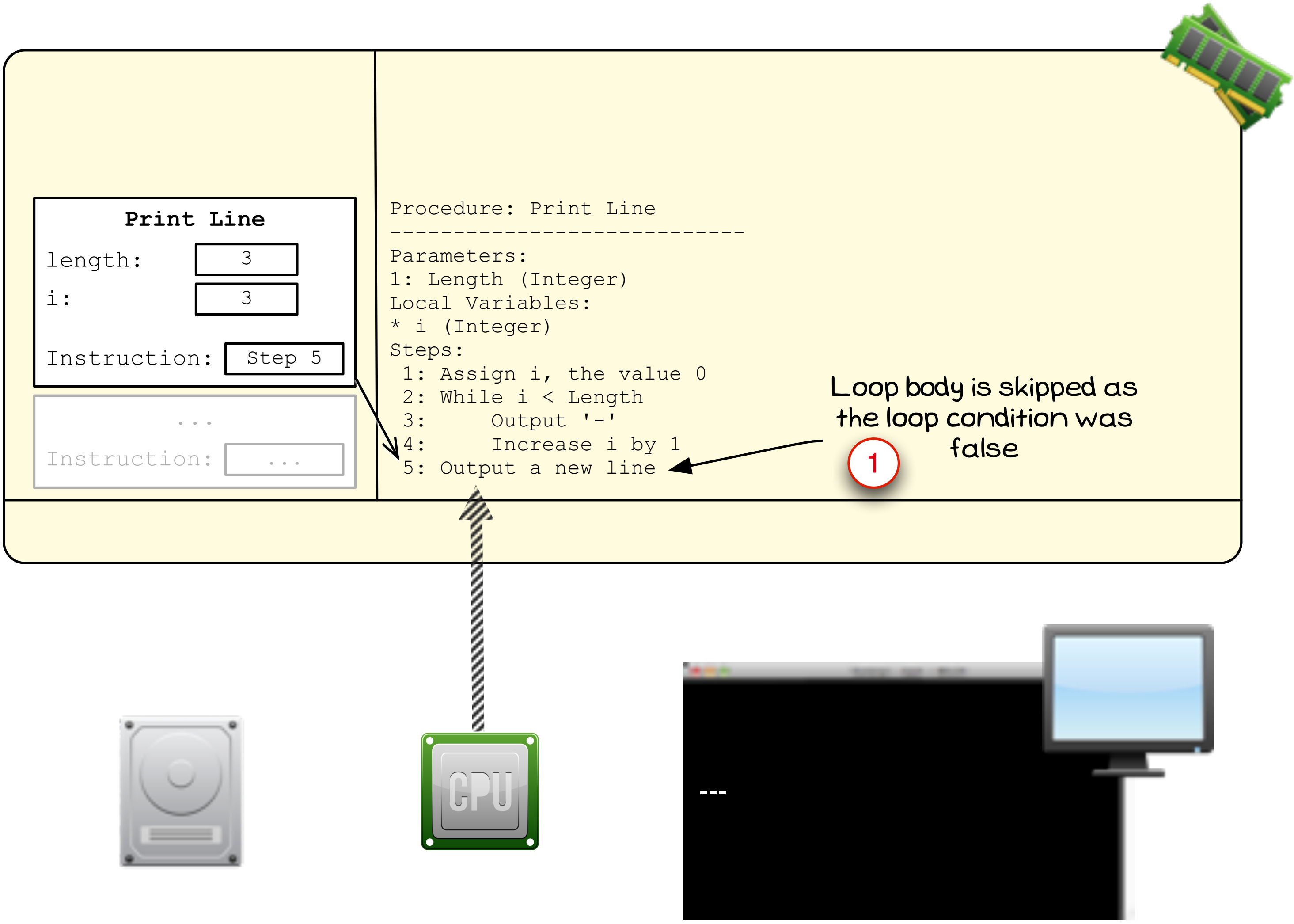
Instruction: ...

```
Procedure: Play Game
----------------------------
Local Variables:
* My Num, Guess Num (Integer)
* Got It (Boolean)
Steps:
 1: Assign My Num, a Random number between 1 and MAX_NUMBER
 2: Assign to Guess Num, the value 0
 3: Output 'I am thinking of a number... 1 and ', and MAX_NUMBER
 4: Repeat
 5:   Increase Guess Num by 1
 6:   Assign Got It, Perform Guess(Guess Num, My Num)
 7: Until Guess Num >= MAX_GUESSES or Got It
 (* While Guess Number < MAX_GUESSES and not Got It *)
 8: If Not Got It then
 9:     Output 'You ran out of guesses... ', and My Num
```

1 Perform Guess is called, and the result returned is assigned to Got It

2 Perform Guess returns false, so this is stored in got it

```
I am think of a number ...
Guess 1: 50
The number is less than 50
Guess 2: 25
The number is larger than 25
```



Print Line

length: 3

i: 3

Instruction: Step 5

...

Instruction: ...

```
Procedure: Print Line
----------------------------
Parameters:
1: Length (Integer)
Local Variables:
* i (Integer)
Steps:
 1: Assign i, the value 0
 2: While i < Length
 3:     Output '-'
 4:     Increase i by 1
 5: Output a new line
```

Loop body is skipped as the loop condition was false

```
---
```