REPETITION

PREPARED BY: LUIS MEING

OBJECTIVES:

01 **Definition**

02 **Parts of a Loop**

03 **Types of Loops**

04 **Repetition Structures in Python**

05 **Loop Nesting**

1.1 Definition

Repetition

- the recurrence (duplication) of an action or event

1.1 Definition

Repetition Control Structures

- repetition control structures are also referred to as iterative structures

- these are groups of code which are designed to repeat a set of statements

1.1 Definition

Repetition Control Structures

- this repetition can repeat zero or more times, until some control value or condition causes the repetition to cease

1.1 Definition

Loops

- most common term to describe repetition control structures

Declaration of ‘counter’ variable

- the creation of a variable that will be compared every time the body is executed at least once

Condition

- the logic that evaluates True or False
- if the condition is True, the loop will repeat
- if the condition is False, the loop will end and exit the structure

Body

- the code block/s within the loop

‘counter’ Variable Update

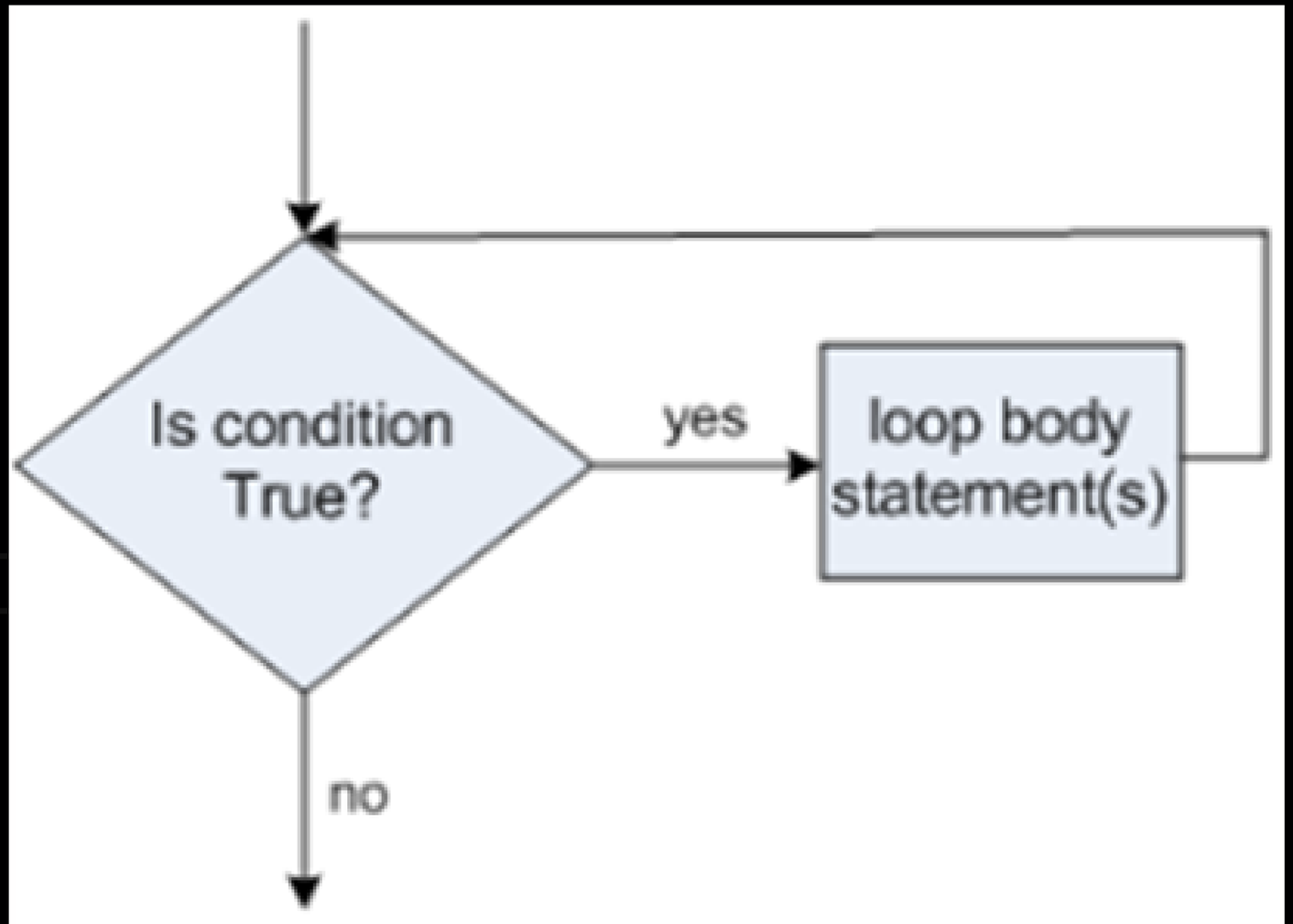
- the change that happens to the ‘count’ variable if the condition is true

- can use arithmetic operators, direct assignment via a conditional structure, or special keywords

Pre-test

- can execute its body a minimum of zero times, if the initial condition evaluates to False

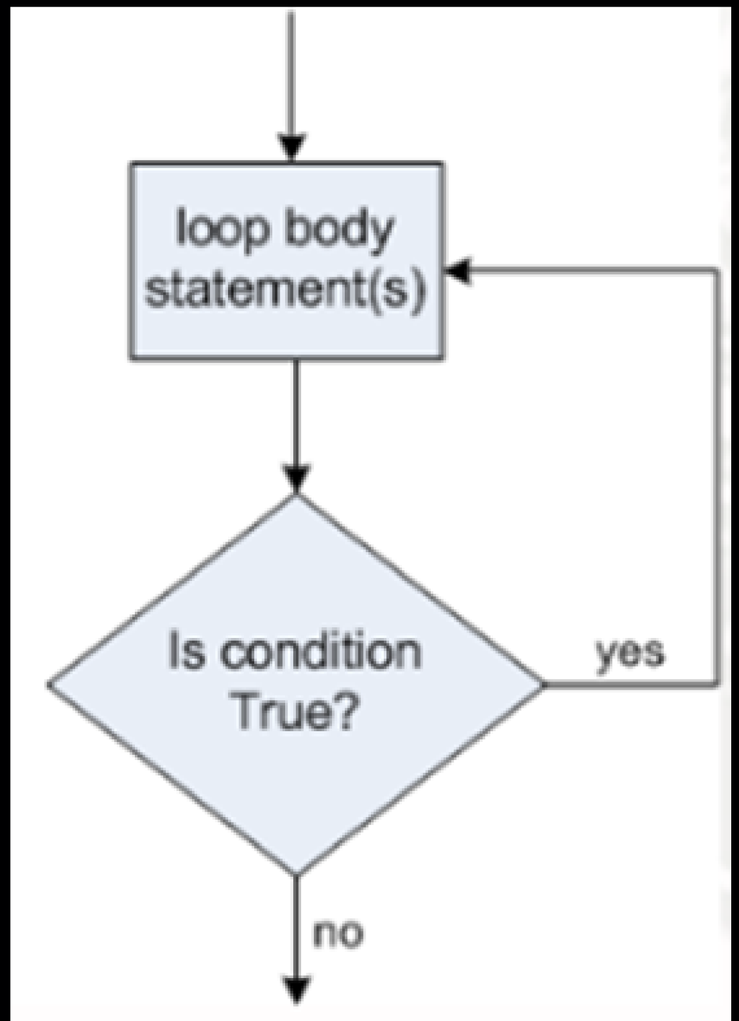
Pre-test



Post-test

- can execute its body a minimum of one time even if the initial condition evaluates to False

Post-test



Post-test

- Python does not have a Post-test loop but an example is Java’s ‘do-while’ Loop

‘while’ Loops

- is a statement in Python that repeatedly executes a block of statements if a test at the top keeps being True

‘while’ Loops

- when the test in the ‘while’ condition becomes False, the process continues to the statement that is below the ‘while’ block

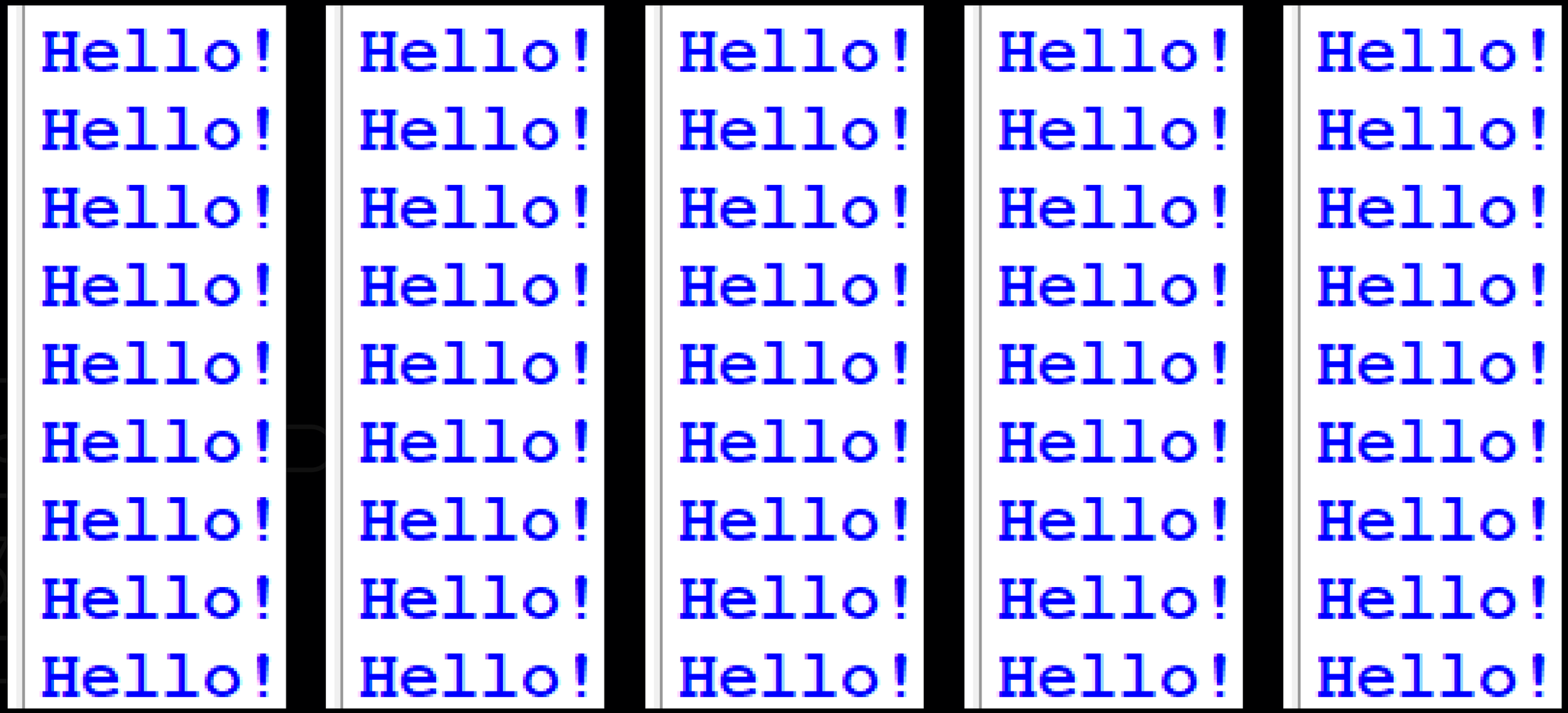
‘while’ Loops syntax

<‘counter’ Variable Update>
while(<condition>)
    <body>
    <‘counter’ Variable Update >

‘while’ Loops syntax

```
1 i = 0
2
3 while i < 5:
4         print("Hello!")
```

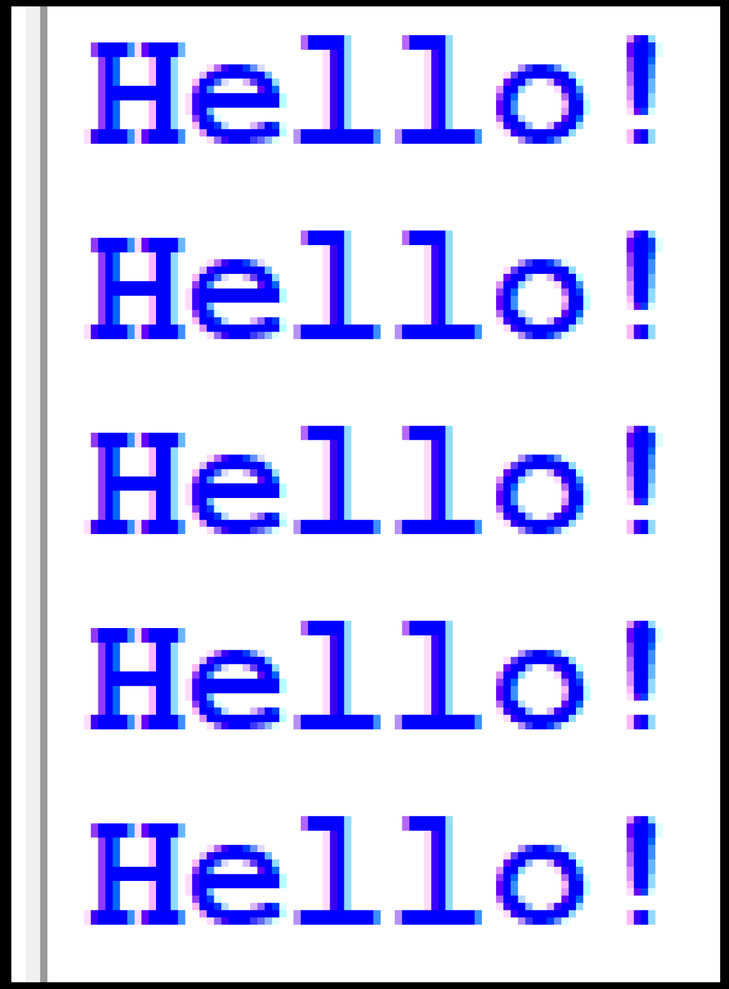
‘while’ Loops syntax



‘while’ Loops syntax

```
1 i = 0
2
3 while i < 5:
4         print("Hello!")
5         i += 1
```

‘while’ Loops syntax



‘while’ Loops syntax

```
1  my_list = [1, 2, 3, 4, 5]
2
3  i=0
4
5  while i < len(my_list):
6      if my_list[i]%2==0:
7          print(my_list[i], "is even")
8      else:
9          print(my_list[i], "is not even")
10     i=i+1
```

‘while’ Loops syntax

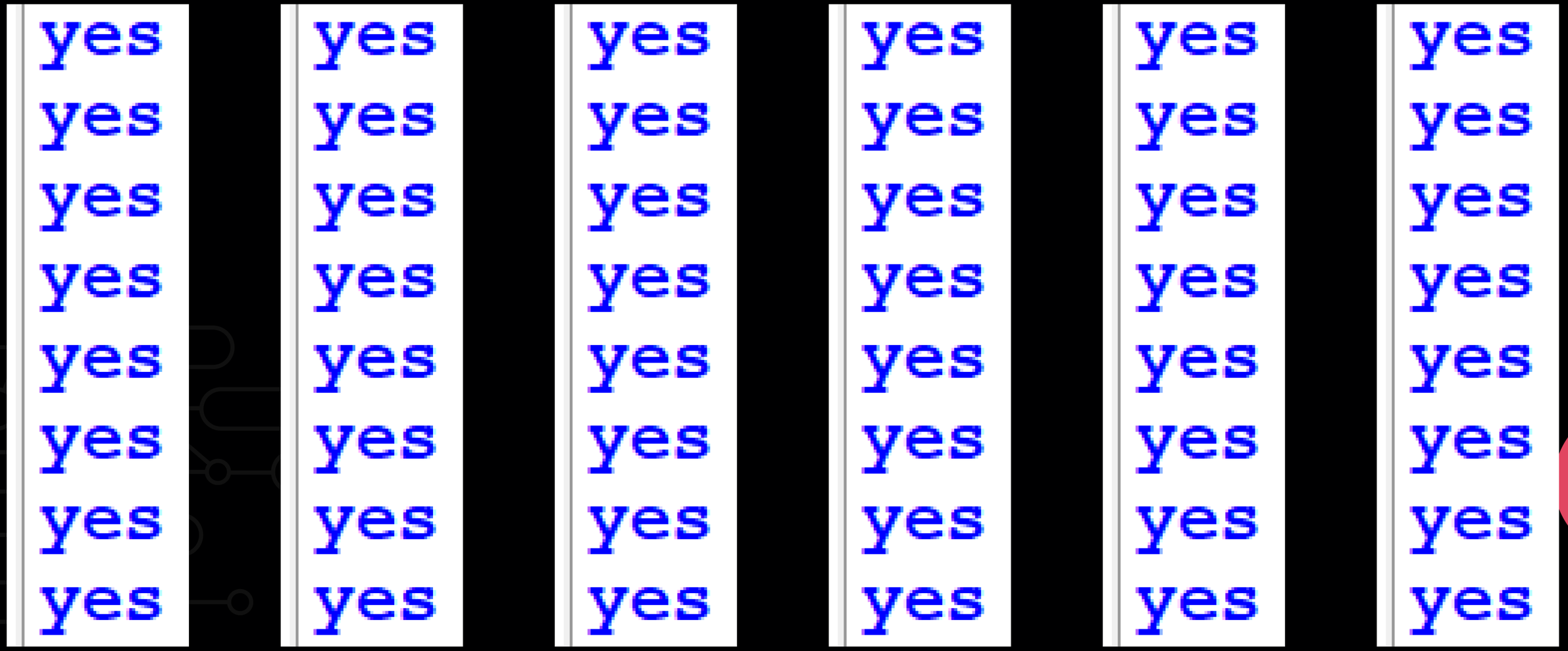
Or

while(<condition>)
    <body>

‘while’ Loops syntax

```
1 while True:
2     print("yes")
```

‘while’ Loops syntax



‘while’ Loops Optional Keywords

- ‘break’ Keyword
  - stops the loop, exits the structure
  - used only inside the loop

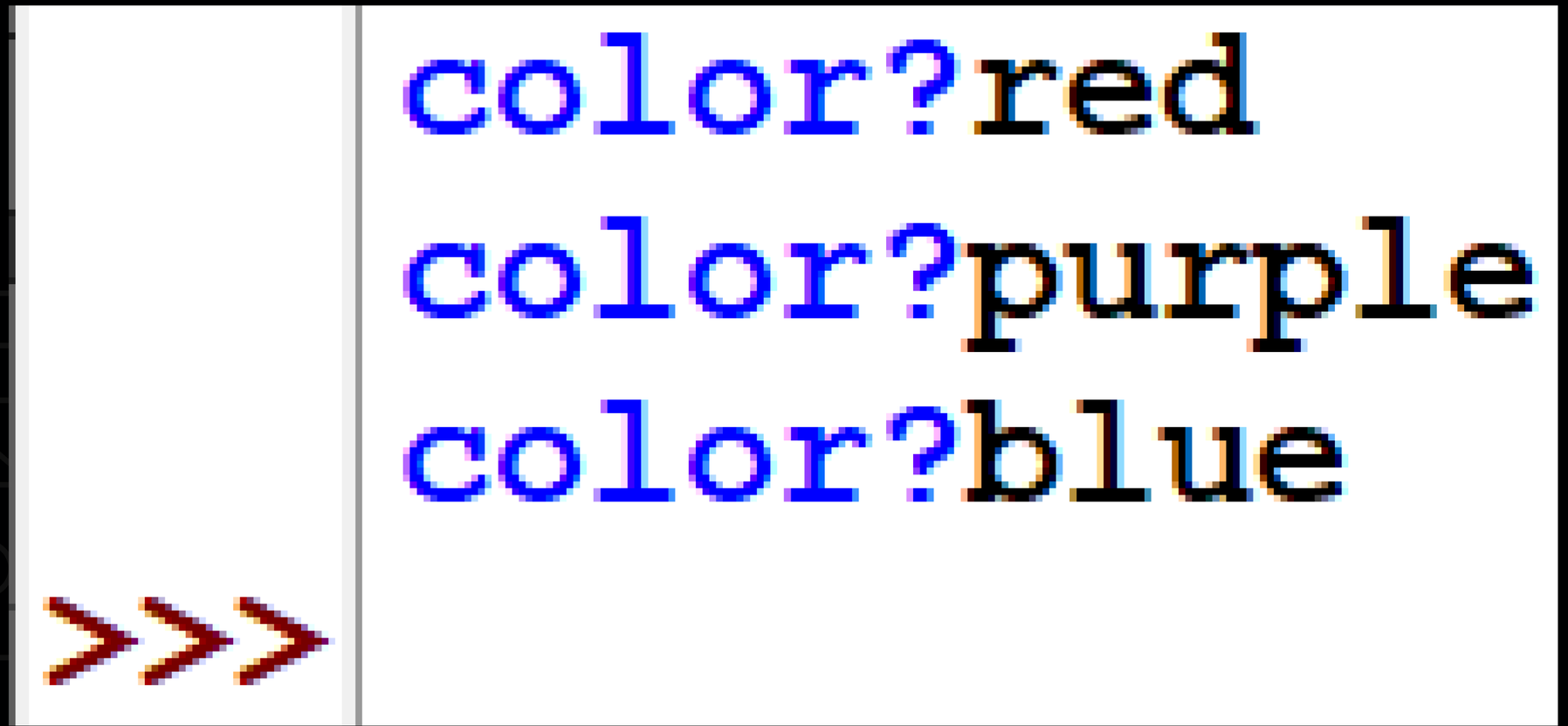
‘while’ Loops Optional Keywords

- ‘break’ Keyword

```
1 while True:
2         response=input("color?")
3         if response=="blue":
4                 break
```

‘while’ Loops Optional Keywords

- ‘break’ Keyword



‘while’ Loops Optional Keywords

- ‘continue’ Keyword
  - goes directly to the top, checking the condition and will execute code again if condition is False
  - used only inside the loop

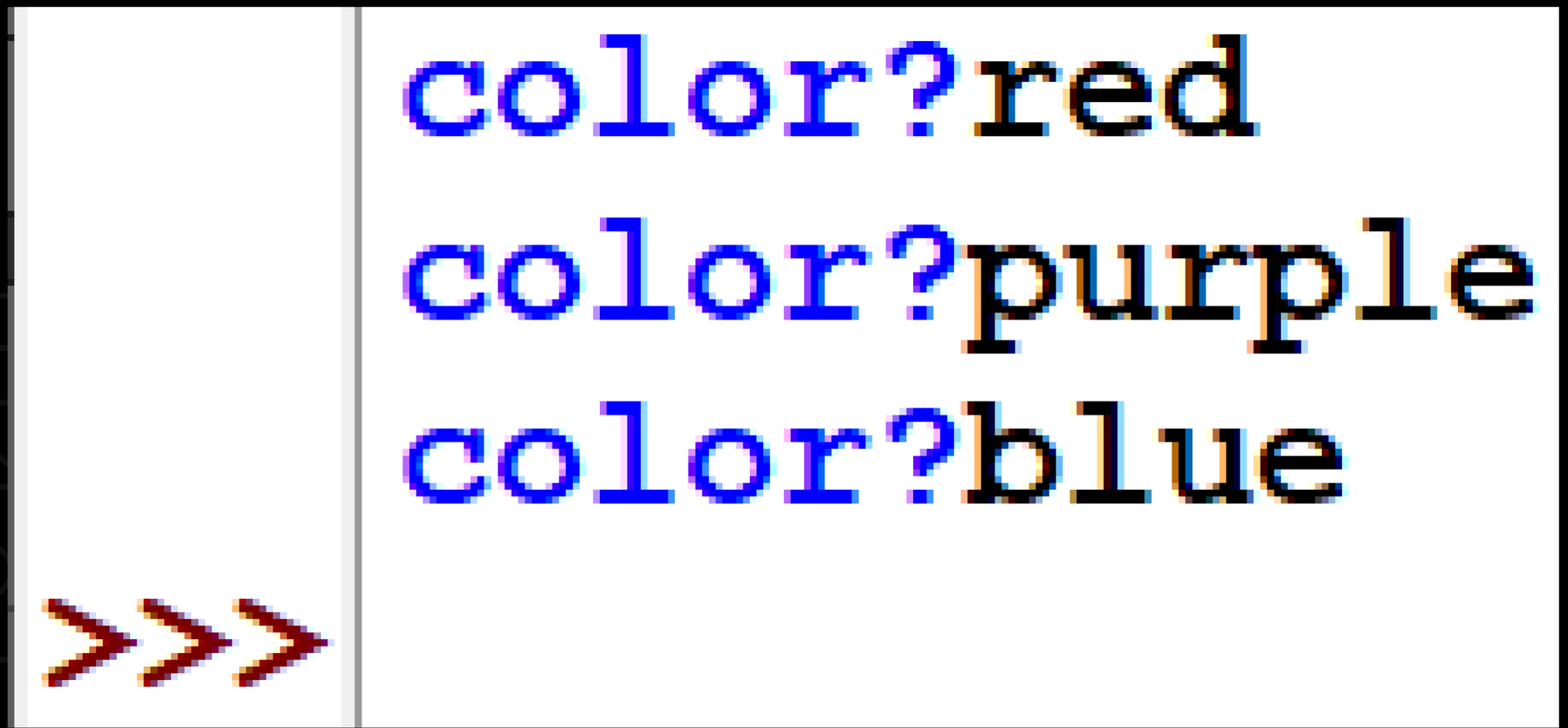
‘while’ Loops Optional Keywords

- ‘continue’ Keyword

```
1 while True:
2     response=input("color?")
3     if response=="blue":
4         break
5     else:
6         continue
7     print("I am ignored")
```

‘while’ Loops Optional Keywords

- ‘continue’ Keyword



‘while’ Loops Optional Keywords

- ‘else’ Keyword
  - this runs if and only if the loop is exited, exited by not triggering a ‘break’ Keyword
  - is used outside the loop indentation

‘while’ Loops Optional Keywords

- ‘else’ Keyword

```
1 i = 0
2 while i<3:
3     response=input("color?")
4     if response=="blue":
5         break
6     i += 1
7 else:
8     print("successful 3 iterations")
```

‘while’ Loops Optional Keywords

- ‘else’ Keyword

```
color?red
color?purple
color?green
successful 3 iterations
```

```
>>> color?blue
```

‘while’ Loops Optional Keywords

- ‘pass’ Keyword
  - literally, does nothing but is used as a substitute to occupy code blocks that require content

‘while’ Loops Optional Keywords

- ‘pass’ Keyword

```
1 i = 0
2 while i<3:
3     response=input("color?")
4     if response=="blue":
5         pass
6     else:
7         print("this is not blue!")
8     i += 1
```

‘while’ Loops Optional Keywords

- ‘pass’ Keyword

```
color?red
this is not blue!
color?green
this is not blue!
color?purple
this is not blue!
>>>
```

```
color?blue
color?blue
color?blue
>>>
```

‘for’ Loops

- usually used to go through sequence structures
- also uses optional keywords ‘break’, ‘continue’, ‘else’, ‘pass’
- automatically increments the counter variable

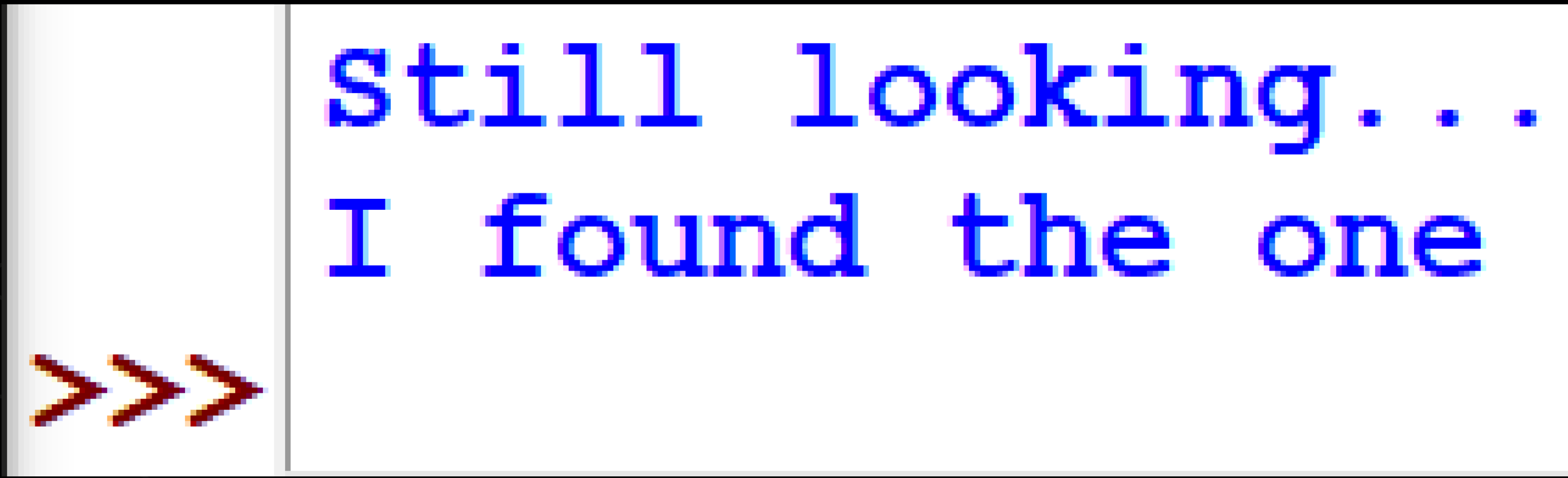
‘for’ Loops syntax

```
for <any_other_var_name> in <structure>:
        <content>
```

‘for’ Loops syntax

```
1 my_list = [1, 2, 3, 4, 5]
2
3 for i in my_list:
4     if i == 2:
5         print("I found the one")
6         break
7     else:
8         print("Still looking...")
```

‘for’ Loops syntax



Loop Nesting

- can use ‘for’ and ‘while’ Loops
- much like nested ‘if’ statements, there can be loops inside other loops

Loop Nesting

- there can be a ‘while’ Loop inside a ‘while’ Loop, a ‘for’ Loop inside a ‘for’ Loop, a ‘for’ Loop inside a ‘while’ Loop, a ‘while’ Loop inside a ‘for’ Loop and so on and so forth.
- take note of indentations, they make a difference between consecutive loops instead of nested loops and vice versa.

Loop Nesting

```
1  my_list = [1, 2, 3, 4, 5]
2
3  i=0
4
5  while i < len(my_list):
6      if my_list[i]%2==0:
7          print(my_list[i], "is even")
8      else:
9          print(my_list[i], "is not even")
10     for x in my_list:
11         print(my_list[i])
12     i=i+1
```