Chapter 4: Loops

4.0.1 Objectives

Repetition is a fundamental programming tool. This chapter introduces three types of loops, which are the basic code building blocks used to repeat actions in a program. At the end of this unit, students should be able to reason about the number of repetitions and terminating conditions of a loop, and they should be able to apply `for`, `while`, and `forever` loops in their programs.

4.0.2 Topic Outline

4.0.3 Key Terms

Control statements: repetition and iteration	Fixed and variable repetitive statements
Terminating condition	Increments
Infinite loops	For loop
While loop	Start value or beginning condition

4.0.4 Key Concepts

Iteration allows a short program to represent a long series of steps by including repeated sequences.

A part of a program that repeats commands is called a **loop**.

Every loop has two parts:

- The **condition** that controls how many times to repeat the loop.
- The **body**, which is a block of code that is repeated as long as the loop is running.

A loop that never stops repeating is called an **infinite loop**. An infinite loop will prevent a program from ever finishing, so usually a program with an infinite loop is not desirable. Within a browser, an infinite loop will even prevent a program from ever responding to mouse clicks, so Pencil Code will try to detect and interrupt programs with infinite loops.

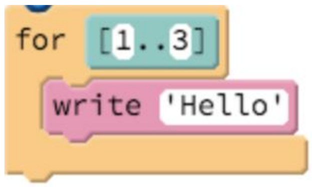
To avoid an infinite loop, the condition that controls how many times the loop is repeated must written correctly. There are two main ways to make a looping condition in CoffeeScript:

- `for` repeats a block of code one for each item in an **iterated list**, and
- `while` repeats a block of code as long as the **loop condition** remains true.

Using Loop Blocks

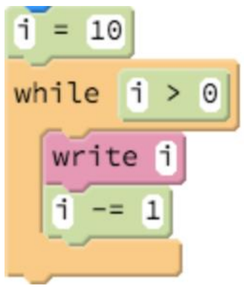
The loop structure is located under the Control palette in the block. There are three types of loop structures available:

`for` loop: Loops over a set of program statements for a fixed number of times in fixed increments. The following loop writes the word “Hello” three times.



```
for [1..3]
  write 'Hello'
```

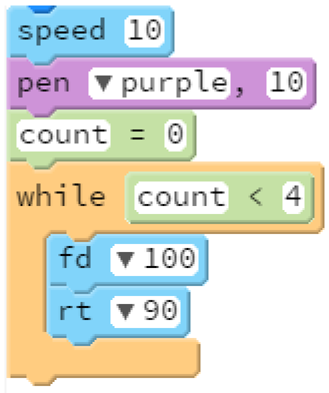
`while`: Loops over a set of program statements as long as the evaluating condition returns true. The actual number of iterations is not known until execution time.



```
i = 10
while i > 0
  write i
  i -= 1
```

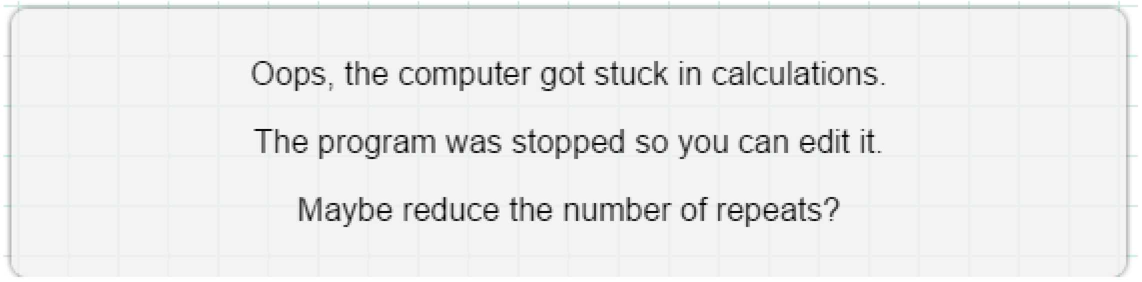
It is possible that, due to a logical error, the loop may be set to iterate through the block of code an infinite number of times. This is very common with beginning programmers. Pencil Code prevents this by generating an error and halting program execution. Internally, Pencil Code keeps a timer when inside a loop. If a program remains stuck inside the loop for several seconds without processing input, Pencil Code assumes the loop is stuck and interrupts execution.

Please see below for an example:



```
speed 10
pen purple, 10
count = 0
while count < 4
  fd 100
  rt 90
```

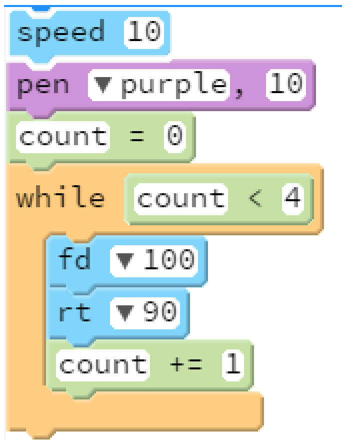
Since count will never reach 4, Pencil Code generates an error after drawing the square.


Oops, the computer got stuck in calculations.

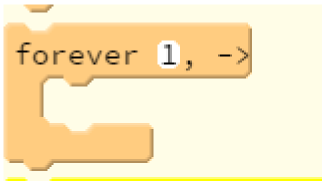
The program was stopped so you can edit it.

Maybe reduce the number of repeats?

The fixed code looks like this:


```
speed 10
pen purple, 10
count = 0
while count < 4
  fd 100
  rt 90
  count += 1
```

`forever`: An ordinary `while` loop that loops forever will cause the browser to **hang**, which means it freezes up the page so the user is unable to interact with it. To repeat a process forever, a program can use a `forever` block which implements a loop by pausing briefly after every iteration to allow the browser to process input. This pause makes the `forever` loop different from the `for` and `while` loops above. The `forever` loop can be used to repeat something indefinitely without freezing the entire browser. The number after `forever` controls the length of the pause: it is the number of times to repeat every second.


```
forever 1, ->
```

Using for Loops in Text

If students have not yet experimented with writing programs in text, this section is an excellent time to suggest that they try out text coding. The examples below will all be shown in text. Several of the ideas in loops are more convenient to work with in text-mode than in block-mode. For example, it is easier to switch a `for` loop into a `while` loop or add or remove an iterator variable in a `for` loop in text-mode than in block-mode.

There are several kinds of `for` loops, but they all work in the same general way, this is, each kind of `for` loop repeats code once for each item in an iterated list. It is easiest to switch between these forms of the `for` loop in text-mode.

`for [1..3]`
`  write 'hello'`
	`for x in [0...3]`
	`  write 'hello', x`
		`for x in ['alice', 'bob', 'carol']`
		`  write 'hello', x`

hello hello hello	hello 0 hello 1 hello 2	hello alice hello bob hello carol

All of these loops repeat their bodies three times because each of them iterates over a list with three elements. The loops that use a variable "`for x in`" assign successive values for x each time around the loop, and the loop body can use that variable to create slightly different output each time.

Here is some information about list notation. (Chapter 8 discusses arrays in greater detail.) A CoffeeScript range array is enclosed in brackets, and `[1..3]` with two dots between integers indicating that the list of integers starts at 1 and ends at (and including) 3.

To indicate a list that does not include the last item, use three dots `[0...3]`. This three-dot form is particularly useful because (unlike the two-dot form) it can express an empty array such as `[3...3]` (a list with no items). An array can also be written explicitly by separating elements using commas or by writing each element on its own line.

Loops using `for` terminate automatically when they get to the end of their list, so it is more difficult to make the mistake of creating an infinite loop using a CoffeeScript `for`. The only way to do so is to create an infinite list to iterate!

Using while loops

A loop made with `while` checks a loop condition. If the loop condition is true, the loop runs the code in the body of the loop then repeats the process, doing another check of the loop condition at the start of each repetition. Since the loop condition must be false for the loop to terminate, it is very easy to mistakenly create an infinite loop by writing a condition that never becomes false.

`j = 0`
`while j < 3`
`  j += 1`
`  write 'checking', j`
`write 'finished'`
	`roll = -1`
	`while roll isnt 1`
	`  roll = random(6)`
	`  wrote 'got', j`
	`write 'finished'`
		`countdown = 3.5`
		`while countdown isnt 0`
		`  countdown -= 1`
		`  write countdown`
		`write 'finished'`

checking 1 checking 2 checking 3 finished	got 4 got 0 got 5 got 5 got 1 finished	2.5 1.5 0.5 -0.5 -1.5 …. (an infinite list of numbers) Oops!

The first two examples terminate, and the third is an example of a buggy program with an infinite loop. In all these cases, the programmer has included three things in the looping program:

- A clear **starting state** (for example, `j = 0`, `flip = -1`, or `countdown = 3.5`).
- A **loop condition** that indicates that the loop should continue repeating. In the programs above, the loop condition repeats the loop when `j < 3`, `flip isnt 1`, or `j isnt 0`.
- A **state change** that eventually leads to the loop condition becoming false. In the programs above, the programs change j, flip, and countdown.

All these programs proceed using **variable assignments** that involve single-equals-assignment operators.

- An assignment such as `j += 1` increases the value of j by one.
- An assignment such as `roll = -1` or `roll = random(6)` replaces the value of the variable roll.
- An assignment such as `countdown -= 1` decreases the value of countdown by one. Notice that the minus before the equals sign means "subtract from this variable and set the value." It is equivalent to `countdown = countdown – 1`.

In these programs, the successive variable assignments change the variables involved in the loop condition. In the first two programs, the assignments eventually lead to the loop condition being false. However, in the third program, the assignments decrease a non-integer countdown by one and they never hit exactly zero, so the loop condition never becomes false. You could fix this loop by changing the loop condition, the starting state, or the decrease amount.

Choosing Between Text-Mode and Block-Mode

In Pencil Code, block code and text code are perfectly equivalent (anything that can be done in one representation of the code can be done in the other) and you can switch between the two modes at any time. The choice between working in text or blocks is a matter of personal productivity.

Blocks are particularly helpful for those who are new to a language. Blocks facilitate the use of correct syntax and help programmers recognize patterns of allowable code. However, using blocks makes it difficult to find choices that are not provided on the palette.

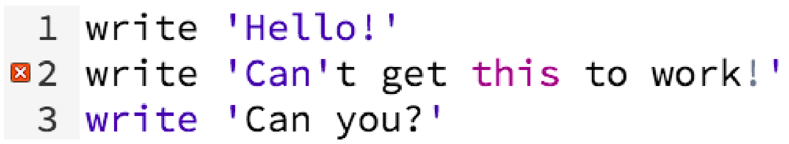
Programming text lets programmers enter a program as fast as they can type. They are not limited by the choices on a palette. Programming in text, however, requires the programmer to remember and follow the syntax rules and it is very easy to create text programs that do not run due to syntax errors.

Although block languages have been improving, professional programmers still write their programs using text because one they are familiar with the programming language, they can work more quickly with text than with blocks.

Finding and Fixing Syntax Errors

Writing text programs is a skill that takes some time to develop. There are two strategies for learning syntax:

1. Observe and copy examples of correct syntax. In Pencil Code, block is useful for this. If students do not recall how a specific syntax works, they can always flip to block to try out something, and then switch back to text once they are confident.
2. Pay attention to syntax errors reported by the computer and fix them right away, before too many syntax errors are created. In Pencil Code, syntax errors are highlighted in text mode with an "x" to the left of the code, like this:



```
write 'Hello!'
write 'Can't get this to work!'
write 'Can you?'
```

The red "x" will sometimes appear when there is an unfinished line of code. However, if the line of code is finished, students should pay attention to the red "x" and try to fix it right away. It is much easier to fix a single syntax error in a program than fixing a program with many syntax errors.

This code snippet is an example of "mismatched quotes". Block mode automatically corrects bare apostrophes by prefixing them with "\" to tell Pencil Code that to include an apostrophe and not to end the string. In text mode, the programmer needs to type this backslash before the apostrophe explicitly.

Because, by definition, a syntax error is a part of the program that the computer does not understand, the computer sometimes puts the “x” in the wrong place. The error in the program might not be spotted until more lines of code are added. The error may be on a line above the “x” and it might have been caused by a problem other than the one the computer indicates.

Common Syntax Errors

Here are some examples of common syntax errors in CoffeeScript. One way to learn the syntax of a programming language is to do a bit of practice fixing examples of syntax errors like this.

Explanation	Example with an error	Fixed example
Mismatched quotes	`write('can't get this to work')`	`write('can\'t get this to work')`
Using word-processor-style “smart quotes”	`write(‘quotes must be straight’)`	`write('quotes must be straight')`
Mismatched parentheses	`write(1+(2*(x - 1))`	`write(1+(2*(x - 1)))`

Missing comma between arguments	`moveto 100 200`
	`           ^`
		`moveto 100, 200`

Misaligned indenting creating incorrect scoping	`if pressed('X')`
	`     fd 100`
	`         rt 90`
	`     ^^^^`
		`if pressed('X')`
		`     fd 100`
		`     rt 90`

Mixing up different brackets	`for x in (1..10)`
	`  write x`
		`for x in [1..10]`
		`  write x`

Programming language syntax is very sensitive to punctuation and, in some cases, spacing. People are usually good at spotting a missing word or a misspelling but it takes practice to pay attention to the punctuation and spaces in a program.

Common Runtime Errors

Here are examples of the most common runtime errors. A program that uses the language syntax correctly may still have runtime errors by referring to a name, variable, or function that has not been defined by the time it is run.

Explanation	Example with an error	Fixed example
Missing quotes	`write hello`	`write 'hello'`

Variable used before definition	`write x * 7`
	`x = 10`
		`x = 10`
		`write x * 7`

Misspelled function name	`wrte 'hello'`	`write 'hello'`

4.1.1 Suggested Timeline: 1 55-minute class period

Instructional Day	Topic
2 Days	Lesson Plan I: For Loops and descending for loops. Practice with for loops
1 Day	Lesson Plan II: Question Bot
1 Day	Lesson Plan III & IV: While loops and forever loops. Practice with while loops
1 Day	Lesson Plan V: Tracing a loop with values

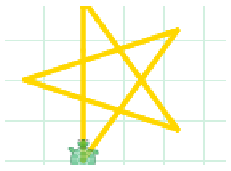
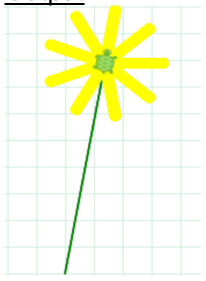
4.1.2 Standards

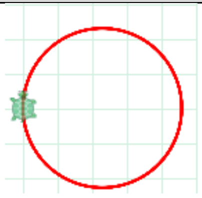
CSTA Standards	CSTA Strand	Learning Objectives Covered
Level 3 A (Grades 9 – 12)	Computational Thinking (CT)	Explain how sequencing, selection, iteration and recursion are building blocks of algorithms.
Level 3 A (Grades 9 – 12)	Computing Practice & Programming (CPP)	Use Application Program Interfaces (APIs) and libraries to facilitate programming solutions.

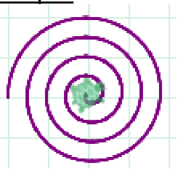
4.1.3 Lesson Plan I

This lesson introduces the 'For Loop' using CoffeeScript. It includes both the ascending and descending variants.
Note: Make sure you are in block mode. Type in the code (switch to block-mode if needed) and click the play arrow to demonstrate the results.

Content details	Teaching Suggestions	Time

Dandelion: Code
`speed 20`
`pen green`
`rt 10`
`fd 200`
`pen yellow, 10`
`for x in [1..9]`
`  fd 50`
`  bk 50`
`  rt 360 / 9`
 	
Gold Star

Loop 360
	Demonstrate for and while loop (Control panel). Type the code as shown and click play to generate the dandelion.
	Point to the students how x takes values from 1 to 9 to draw 9 petals.
	Explain that `rt` moves the turtle by an angle for every iteration. Give students a chance to try this out.
	Output
	
	You could also have the students experiment with Gold Star and Loop 360 as they are good examples for reinforcing the same concept. Click on the link below for the source code for these two examples.
	http://guide.pencilcode.net/edit/loops/
		Demonstration: 20 minutes
		
		Student Practice:

Content details	Teaching Suggestions	Time
		30 minutes

Spiral: Code
`pen purple, 10`
`for x in [50...1]by -1`
`  rt 30, x`
 	
Output

	Once students understand the for loop ascending, introduce the for loop variant for descending loops. The loop decreases from maximum value to minimum value in regular negative intervals. For example:
	
	`for x in [50..1] by -1`
	
	Explain how the angle of 50 is decremented by 1 for every iteration of the loop.
	Click on the link below for the source code for this example.
	http://teachersguide.pencilcode.net/edit/chapter4/spiral
		Demonstration: 10 minutes

4.1.4 Lesson Plan II

Pencil Code allows programmers to move easily between block- mode and text-mode. This lesson introduces programming using text-mode.

Content details	Teaching Suggestions	Time

Code:
`for x in [0...3]`
` write 'hello', x`
 	
Output:
hello 0
hello 1
hello 2
	While in the Pencil Code environment, switch from block-mode to text-mode. Type the program as shown in the left-most column.
	
	Show students the three dots in the loop and explain that the variable x takes on the values 0, 1, 2 and 3 and for each value of x, the word “hello” is printed and the value of x is displayed.
		Demonstration: 20 minutes
		
		Student practice: 80 minutes

(see table below)	Introduce the concept of tracing variables. Have students draw a table and trace the values x takes as shown in the left column.	

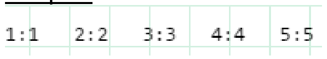
Iteration #	Value of x	write x
1	0	0
2	1	1
3	2	2

Content details	Teaching Suggestions	Time

Code:

```
count = 1
for x in [1..5]
   type count + ':  '  x
   count += 1
```

Output:



```
1:1   2:2   3:3   4:4   5:5
```

Iteration #	Value of x	type count	Count +1
1	1	1	2
2	2	2	3
3	3	3	4
4	4	4	5
5	5	5	6

Demonstrate a simple math operation such as

```
count = count + 1
```

then type the code as shown.

Ask students to trace the code. The tracing values are also shown in the left-most column.

Once the students have worked on it show the solution given here.

As a way of giving students additional practice, provide the following for loops problems and ask them to trace the values.

```
for x in [0..5]
   type count + ':  '  x
   count += 1
for x in [5..1]
   type count + ':  '  x
   count += 1
for x in [0..0]
   type count + ':  '  x
   count += 1
```

4.1.4 Lesson Plan III

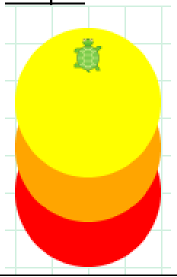
This lesson introduces different for loop that iterates over a collection of data.

Content details	Teaching Suggestions	Time

Code

```
for color in [red, orange,
yellow]
  dot color, 100
  fd 30
```

Output



Demonstrate this program while explaining to that this is a different kind of loop.

The variable ‘`color`’ goes through the collection of colors and executes the ‘dot’ command with that value of color.

The execution of the code traces how ‘dot’ command takes various values.

Encourage students to use variations of this loop to produce interesting output.

Demonstration: 15 minutes
Student Practice: 20 minutes

4.1.5 Lesson Plan IV

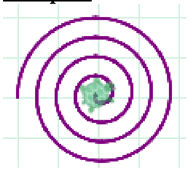
This lesson introduces the first large programming assignment and the idea of iterating on a program over time, in this case the Question Bot program.

Teaching Suggestions	Time
Spiral Assignment: Ask students to use the Question Bot program they created in the previous chapter. Give the students a chance to explore and experiment. Put them in groups and challenge them to develop the most creative Question Bot they can. If they choose to add constructs that have not been covered, allow them to do this but adjust the time accordingly and be quick to move on to the next topic when your class is ready.	Student practice: 55 minutes

4.1.6 Lesson Plan V:

This lesson introduces the While Loop

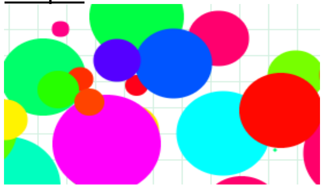
Content details	Teaching Suggestions	Time

Spiral using While Loop
Code
`pen purple, 10`
`x = 50`
`while x > 0`
`  rt 30, x`
`  x = x-1`
 	
Output

	While Loops:
	Type the code as shown for creating a while loop.
	Explain that a while loop is written differently than a 'for loop' but it produces the same output. This helps students understand that there are more than one solution to a problem.
	
	Here is another example of while loop that uses input statements in the code and solves a problem that cannot be solved with a 'for loop'. The program waits for an input from the user to determine if the code within the loop should be executed one more time.
		Demonstration:
		25 minutes
		
		Students Practice:
		All students to us the remainder of the class period and the entire next class period.
		
		Note: Gauge how well the students have understood the content. If feasible, have them also complete lesson plan VI.

4.1.7 Lesson Plan VI

This lesson introduces the 'Forever' loop[MU1].

Content details	Teaching Suggestions	Time

Code
`forever 1, ->`
`  dot (random color), random`
`100`
`  moveto (random position),`
`random position`
	Forever (x) loop:
	Demonstrate the use of this loop using this code. (Confetti)
	Output
	
	Ask students for ideas on where it would be appropriate to use the forever loop.
		Demonstration:
		20 minutes
		
		Students Practice:
		Allow the remainder of the class period and the next full class period.

4.2 Resources

Book: book.pencilcode.net