Topic 4.1: Iteration – The `while` Loop

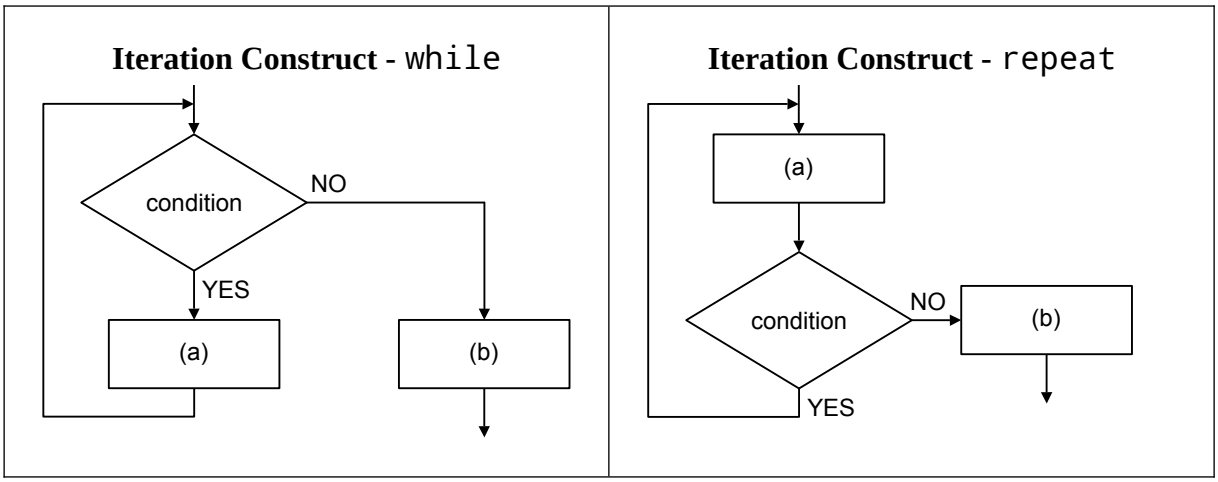
This chapter introduces the `while` loop, Python's simplest and most flexible tool for repetition.

Review of the Iteration Construct

Programs use iteration when a task needs repeating – either a known number of times, or until some condition changes. There are two fundamental types of loop, distinguished by when the condition is checked.

In the first type, the condition is checked before each iteration. In Pearson pseudocode and Java, this is the `WHILE` loop and `while` loop, respectively. If the condition is not true when the loop is reached, the block of code inside the loop will not execute at all.

The diagram below left shows the flowchart diagram of a `while` loop. If the `condition` in the decision block is true, the code block inside the loop, labelled `(a)`, is executed. After execution, the `condition` is checked again, and if found to be `True`, code block `(a)` is executed again. This repeats until the condition in the decision block becomes `False`., at which point, the code block labelled `(b)` begins execution.



The second type of iteration is where the condition is checked after executing the repeating code block exactly once. In Pearson pseudocode and Java, this is the `REPEAT .. UNTIL` loop and `do .. while` loop, respectively.

The diagram above right shows the flowchart diagram of a repeat loop. First, code block `(a)` is executed, and only then the condition is checked. If the condition is true, code block `(a)` will be repeated, and if not, the code block labelled `(b)` begins execution.

The important things to remember about while and repeat loops are:

- The condition of a **while** loop is checked before the code block within the loop executes
- The code block within a **while** loop may execute zero or more times.
- The condition of a **repeat** loop is not checked until the code block has executed one time.
- The code block within a **repeat** loop will execute at least once, but may execute more times.
- For both **while** and **repeat** loops, the flowchart diagram shows the flow returning to a point before the decision block.

Ugly Hybrid `while-repeat` Loops

A word of caution when designing your algorithm using a flowchart: if you place a process block (a code block) before the condition check and another process block after the condition check, both within the loop, it does not always cleanly translate into high-level language code.

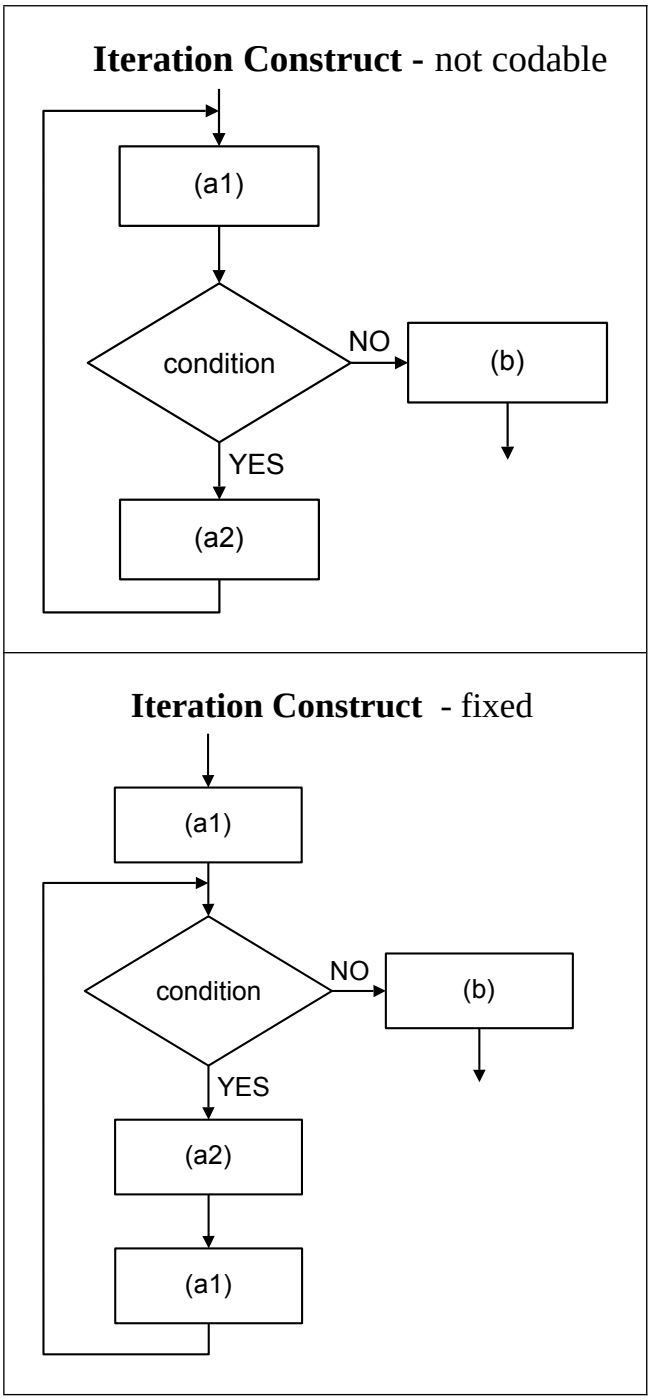
The flowchart to the right shows an example of this. The code in block `(a1)` is within the loop before the condition is checked, while the code block `(a2)` is within the loop after the condition is checked.

There are different ways to implement this in a high-level language one way, show to the right, is to repeat the code block `(a1)` within the loop after code block (a2).

Another way to implement it (not shown diagrammatically) is to add a condition within a repeat loop such that `(a2)` is only executed after the first time through the code block. **However, it is a mortal sin in programming to repeat code, and your future self will be most displeased with you, perhaps even curse you, when they need to look back on your code!**

A third way to implement it (again, not shown) is with an infinite loop with a conditional break out of the loop

In most cases, this type of messy structure can be avoided by some careful consideration of the algorithm.



Python Iteration – The `while` Loop Syntax

The python code for a `while` loop that prints out the numbers from 1 to 5 is given to the right, and below that the equivalent Java and Pearson pseudocode. The syntax is as follows:

- The line starts with the keyword `while`.
- Next, there is the condition, followed by a colon (:).
- The block of code that is to be repeated is indented, usually four spaces.
- The code outside the block has its indentation inline with the line containing the `while` keyword.

Note the similarity to an `if` statement.

The trace table for the code to the left is given below.

```
# Python code
number = 1
while number <= 5:
    print(number)
    number += 1
print('complete')
```

```
// Java code
int number = 1;
while(number <= 5) {
    System.out.println(number);
    number++;
}
System.out.println("complete");
```

```
# Pearson pseudocode
SET number TO 1
WHILE number <= 5 DO
    SEND number TO DISPLAY
    SET number TO number + 1
END WHILE
SEND "complete" TO DISPLAY
```

number	number <=5	Action
1	True	print 1, then number is set to 2
2	True	print 2, then number is set to 3
3	True	print 3, then number is set to 4
4	True	print 4, then number is set to 5
5	True	print 5, then number is set to 6
6	False	loop ends.

Avoiding an Infinite Loop

```
# Python code
number = 1
while number <= 100:
    print(number)
    # forgot: number += 1
print('complete')
```

Examine the code to the right. What will the output be?

Since the code that increments the value stored in `number` has been commented out, with each iteration of the loop, `number` will always equal `1`.

The code will repeatedly print the number `1` and not stop until the program is manually killed. This is referred to as an ***infinite loop***. To avoid this type of error, review any loop you have written to ensure you have:

- initialized the loop variable(s) before the loop
- written the terminating condition correctly
- updated the loop variable(s) inside of the loop such that the condition will eventually evaluate to `False`.

Exercise

Write a Python program that uses a `while` loop to print out a count down from `5 down` to `0`. The output can print each number on its own line, or print all numbers on a single line, separated by a space. Example output

```
10 9 8 7 6 5 4 3 2 1 0
```

`break` and `continue`

These two keywords allow you to prematurely stop executing the code of the current iteration of the loop, and either:

- `break` out of the loop, or
- `continue` with the next iteration.

Examples will make their use more clear.

When `break` is encountered within a loop, the loop exits immediately, regardless of the loop condition. Study the following code and its corresponding output.

```
1  Numbers = [ 3, 5, 7, 3, 8, 9, 2 ]
2  target = 8
3  index = 0
4  while index < len(numbers):
5      if numbers[index] == target:
6          break
7      index = index + 1
8  print("Found", target, "at index", index)
```

```
1  Found 8 at index 4
```

The code above implements a linear search algorithm. Once the index of the target element is found, there is no reason to continue the loop until the end of the list, so `break` is used to stop the iteration. Once `break` is reached, execution of the loop code stops, and execution resumes at the line of code immediately following the loop, line 8 in the above example.

When continue is encountered within a loop, the loop stops execution of the current iteration of the loop, returns to the start of the loop, and performs the condition check to determine whether the loop should be entered again. Study the following code and its corresponding output.

```
1  n = 0
2  while n < 9:
3      n += 1
4      if n % 3 == 0:
5          continue
6      print(n, end=" ")
7  print()
```

```
1  1 2 4 5 7 8
```

Inside the loop of the code above, the value of `n` loops through the printing the numbers from 1 to 9, but the `if` statement containing `continue` will cause any number divisible by 3 to be skipped.

Note: use `break` and `continue` sparingly, as they can often obscure the intent of the code.

Implementing REPEAT Loops in Python

Pearson pseudocode has a `REPEAT` loop, which we have discussed earlier in the section on programming constructs. This loop guarantees that the loop body runs at least once because the condition is checked after the body. Perhaps most programming languages, including C, Java, JavaScript, include a `REPEAT` loop in the language. In Java, you may recall it is the `do..while` loop.

The Python language does not include a `REPEAT` loop. The idiomatic Python replacement that implements the functionality of a `REPEAT` loop is a `while True` loop with a selection statement at the end of the body that includes an explicit `break` to exit the loop. Compare the example Python code and the equivalent Java.

```
// Java example code
do {
   password = getPassword();
} while (!isValid(password));
```

```
# Python equivalent code
while True:
   password = input("Enter password: ")
   if is_valid(password):
      break
```

Exercise

Write a program that:

- repeatedly prompts the user to enter an integer
- stores each entered number in a list
- stops the input of numbers when the user enters the number 0 (zero)
- prints the resulting list
- prints the sum of the numbers in the list

For example:

```
Enter the next number for the list: 5
Enter the next number for the list: 3
Enter the next number for the list: 1
Enter the next number for the list: 0
[5, 3, 1]
Sum: 9
```

`while` versus `for` loops

Like many other high-level programming languages, Python also offers a `for` loop, which we will discuss shortly. Any iteration can be implemented with either form of loop, so the choice comes down to which one is cleaner and more understandable when read. As a rule of thumb, the `for` loop is the better choice when the number of iterations is known – for example traversing a list, string, or range of numbers – and the `while` loop is nicer for an indeterminant number of iterations, such as for input validation, a game loop, input from the user or device, and so on.

Summary

These are some key points to remember about iteration using a `while` loop:

- A `while` loop repeats an indented block as long as its condition is `True`.
- The condition is checked before each iteration, including the first.
- Ensure the loop body changes something that will eventually make the condition `False`; otherwise you get an *infinite loop*.
- Common patterns: input validation, accumulation (sum, count, string building), and nested loops for grids or tables.
- `break` exits a loop early; `continue` skips the rest of the current iteration.
- Choose `while` when the number of repetitions isn't known in advance; choose `for` for definite iteration over a sequence.