CHAPTER 5

LOOPING STATEMENTS

LEARNING OBJECTIVES

After completing this chapter, the reader will be able to :

- Understand the process of stepwise refinement using repetition or iteration structures.
- Appreciate the importance of using control structures effectively, in producing understandable, debuggable, maintainable programs, and more likely to work correctly on the first try.

- Understand the built-in functions for looping.
- Learn to effectively use while loop, for loop, nested loops, and loop control statements.
- Use and implement the while and for repetition structures to execute statements in a program repeatedly.
- Learn to use break statement, continue statement, and pass statement effectively

5.1 Introduction to Looping

- **Loop** is a block of code that is recurring for a stated number of times until some condition is encountered.
- In a ***loop's body*** the code, placed inside the loops, are carried out repeatedly**.**
- ***Example:*** a code written to convert the temperature from degrees Celsius to Fahrenheit

1.2 Purpose of loops

- **“Programming”** involves giving the computer the instructions and data it needs on time.
- *Programming Mechanism*-The process of programming is done in phases, such as composing problem statements, drawing flowcharts, designing coding algorithms, writing a computer program, analyzing and evaluating software, technical writing, keeping software up to date, and so on.

The benefits of using loops in Python are as follows:

- Loops aid in escalating the code reusability.
- Loops simplify complex problems into simple ones, which are very easy to handle.
- Loops aid in easily crisscrossing the elements placed in arrays and linked lists.
- Loops aid the developers to remove the drudgery of physical writing, and implement the upfront and small instructions repetitively.

Classification of flow controls

- Iteration is based on a counter (or *counter loop*) or a condition (or *condition loop*).
- loops are divided into *finite loops (limited)* and *infinite loops (unlimited)*.
- The finite loops are further typified as definite and indefinite loops based on the clarity in defining the number of iterations.

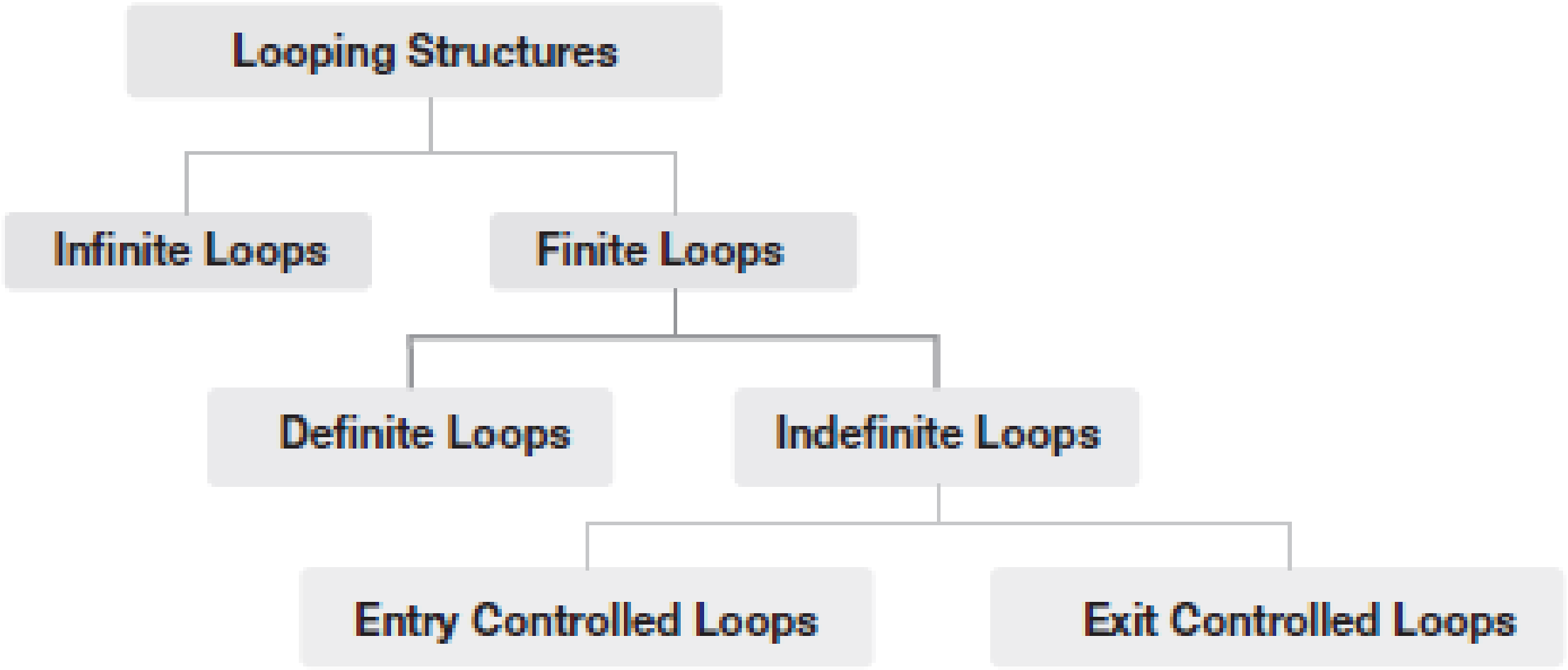


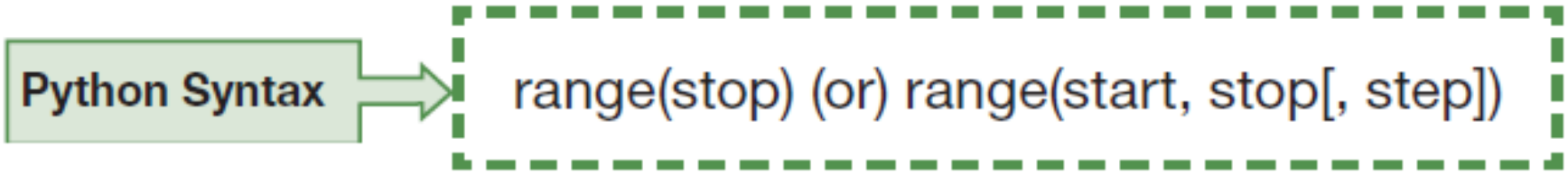
Fig. 5.1: Python Programming – Classification of Flow Control.

- The constructs in an indefinite loop are further subdivided into *pre-test loops or pre-checking loops or entry-controlled loops* and *post-checking loops, or exit-controlled loops*.
- In an *entry-controlled loop*, the condition is tested before the commencement of the loop. In an *exit-controlled loop*, the condition is tested after the commencement of the loop.

5.2 Python built-in functions for looping

- Python has four functions, namely, User-Defined Functions (UDF), anonymous or Lambda functions, recursive functions, *and Built-in functions*.
- *Built-in functions* are created or defined in the programming framework and are always available.
- Python 3.X has 68 built-in functions

range () function



The parameters of the range function are start, stop, and step.

- **start** is an integer; beginning of the sequence of integers that must be returned.
- **stop** is an integer that is fixed beforehand and indicates the last number in the sequence of integers that must be returned.
- **step** is an integer value that limits the increment amongst each integer in the sequence.

The range() function is typically put to two main uses:

[i] Iterating through a for-code loop's body a given number of times.

[ii] Iterating integers faster than with lists or tuples.

5.3.1 while Loop

- Python while loop is a *condition-controlled loop* that supports repeated execution of a statement or block of statements controlled by a conditional expression.

Syntax of while Statement:

The following is a syntactical representation of the while statement:

```
while (condition is true)
    statements detail the steps to perform
    # "some steps" must finally result in the condition being false
```

while	Reserved word used to begin the statement.
condition	Boolean expression determines whether to execute the loop body.
:	colon (:) must follow the condition.
Block	If the while condition is true, one or more statements will be executed.

Note:

Statements in a block must be indented with the same number of spaces from the left. It's called the statement's body.

while loop with else:

When a while statement's condition evaluates to false, any else information (optional) that follows it can be used to define how the processing of the code should proceed.

The *else block* is a block of one or more statements to be completed. It must be indented more spaces.

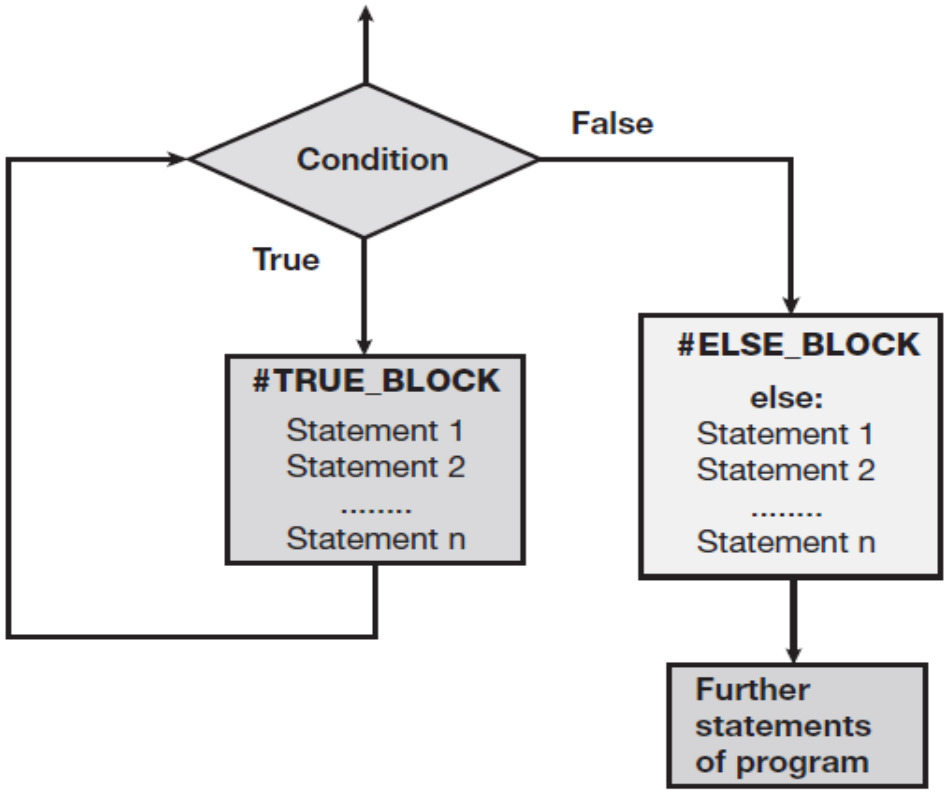


Fig. 5.3: A Flow Chart Depicting the Process of a While Else Loop.

- **Syntax of while loop with else clause:** The following is a syntactical representation of the while loop with the else statement.

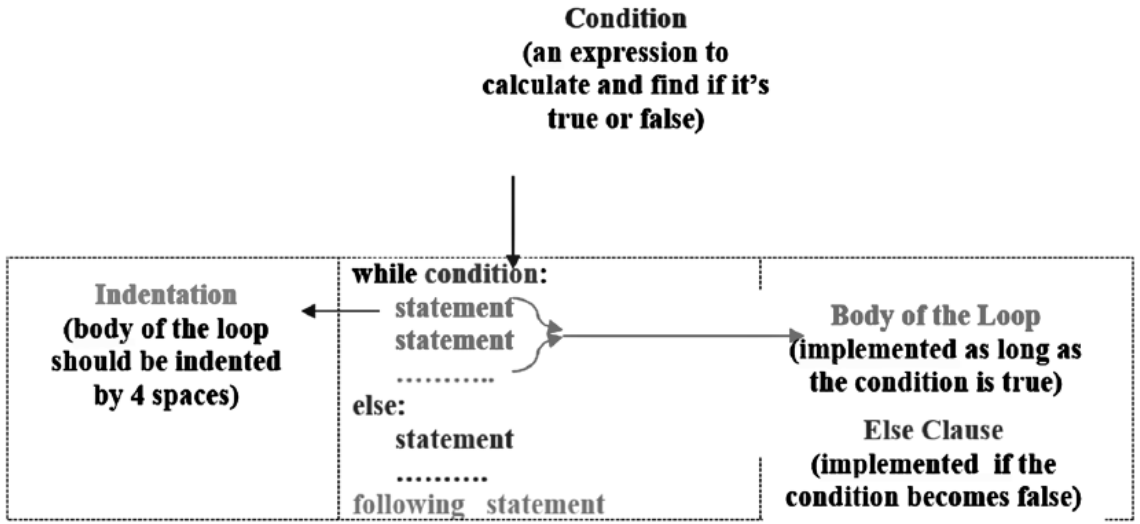


Fig. 5.4: Syntax of while-else Loop in Python

- **while** A reserved word is used to begin the statement.
- **condition** the Boolean expression determines whether or not the body will be executed.
- **:** colon (:) must follow the condition.
- **Block** If the while condition is true, one or more statements will be executed.
- **else** The reserved word else begins the second part of the while/else statement. The else block is a block of one or more statements to be executed if the condition is false. It must be indented with more spaces.

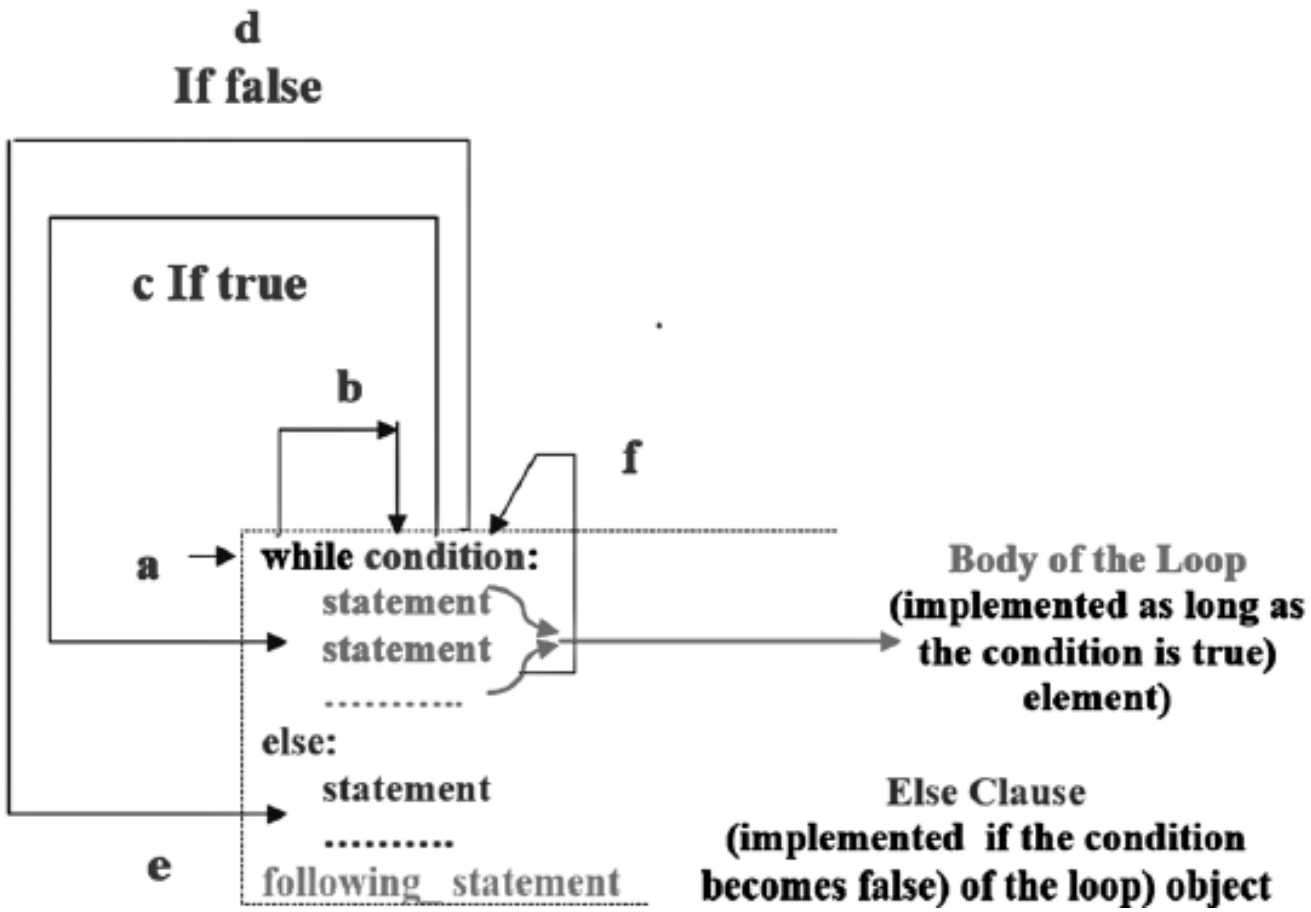
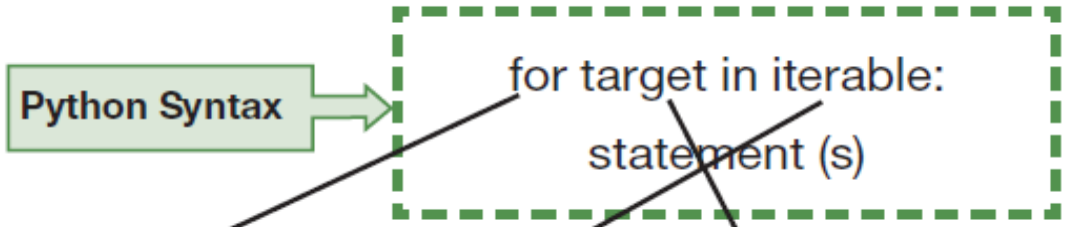


Fig. 5.5: Depiction of the workflow of a simple while–else loop in Python.

5.3.2 for Loop

- **Syntax of *for*** statement: The following is a syntactical representation of the for loop.



It starts with the **keyword "for"** followed by an arbitrary **variable (var)** name. The variable name holds the values of the following **sequence object(iterable)**, which is stepped through the following syntax of the for statement as follows:

structure of the syntax (Fig. 5.6) shows that

for	the reserved word is used to begin the for Loop.
var	variable
iterable or [iterating variable]	object used in iteration. can either be an adjective or a noun .The iterating variable is an individual item during every iteration and is assigned a provisional random variable name.

in	splits the picked item separately from the other(s).
:	colon (:) directs to follow the body of the code that ensues.
Block	Has one or more statements to be executed that need to be undertaken and repeated.

Note:

Statements in a block must be indented the same number of spaces from the left. It's called the statement's body.

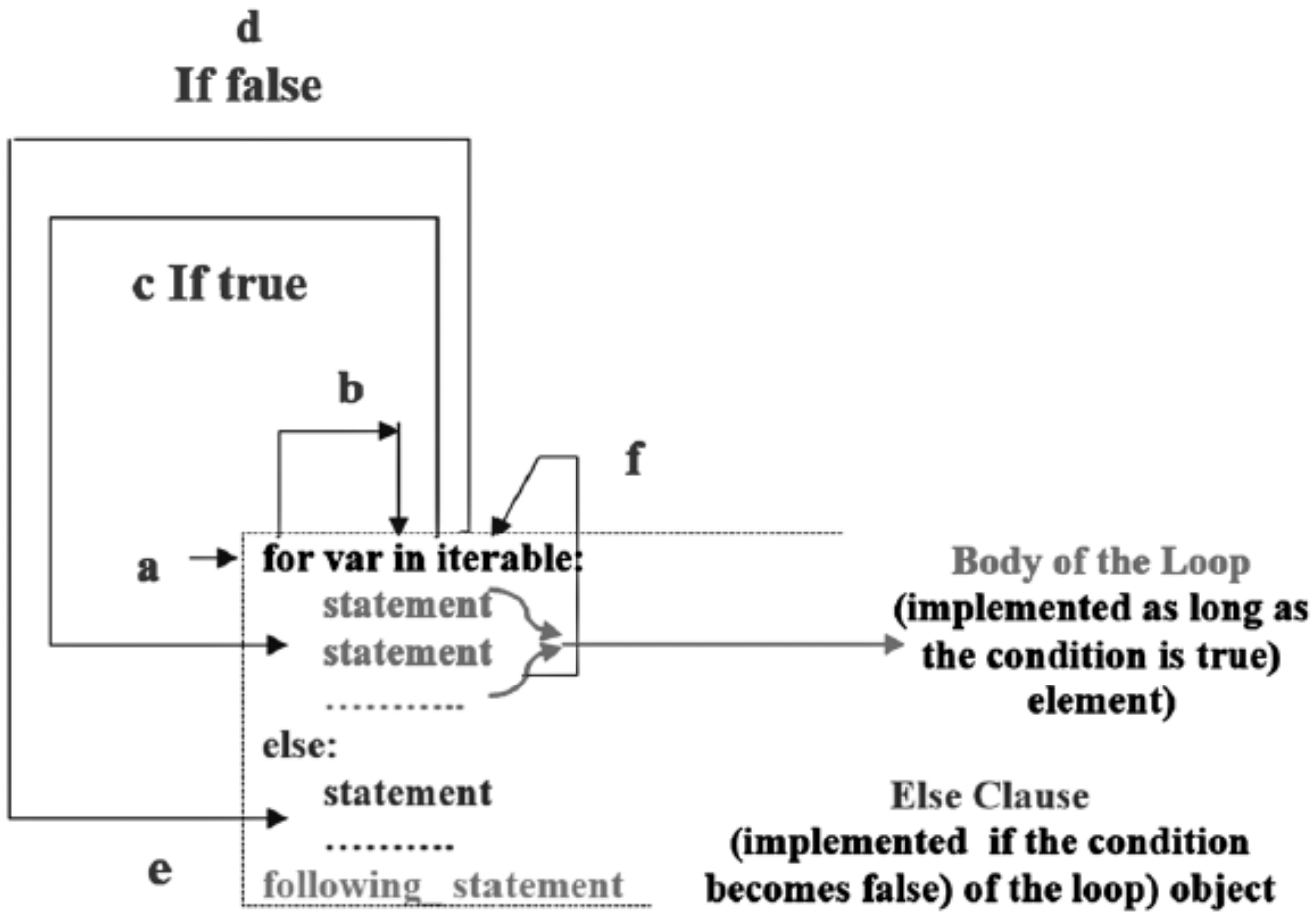


Fig. 5.7: Depiction of the Work Flow of a Simple For else Loop in Python

The flow of an implementation of a for-loop (Fig. 5.7) is as follows:

- The loop is activated with the keyword. After testing the membership expression, it demands the next element or item from the iterable (a collection of elements or items).
- If the iterable is blank, exit the for-loop without running its body.

- If the iterable did produce an element, allocate that element to <var>. If <var> was not previously delineated, it becomes delineated).
- Implement the bounded body of code equal to the number of items or elements in the collection.
- Return to the first line.

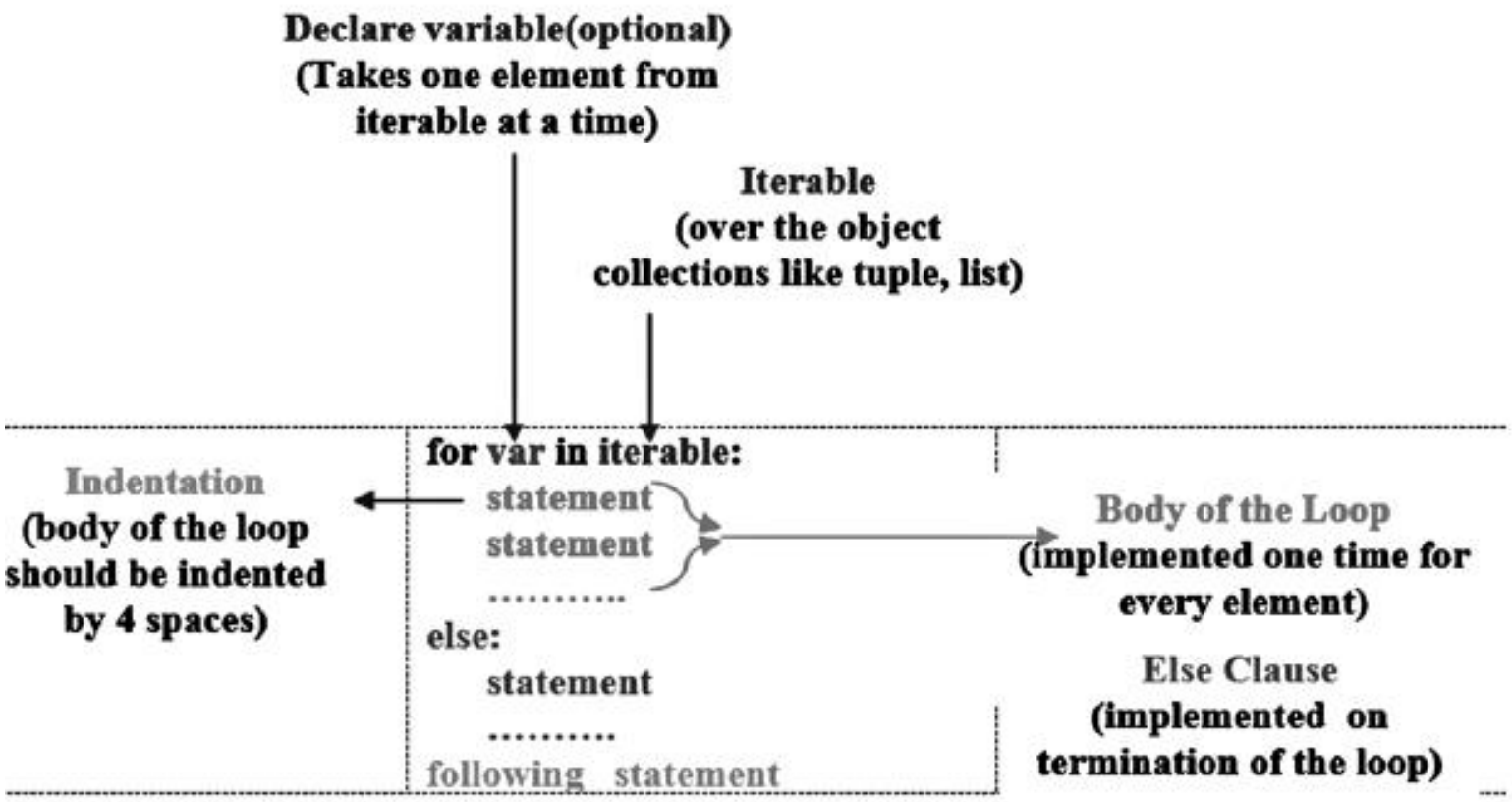


Fig. 5.8: Syntax of for else Loop in Python.

structure of the syntax (Fig. 5.8) shows that

for	Reserved word is used to begin the For Loop.
var	variable
iterable	the object used in iteration.
in	splits the picked item separately from the other(s).
:	colon (:) directs to follow the body of the code that ensues.
Block	Has one or more statements to be executed that need to be repeated.
else	reserved. else block executes statements if the condition is false.
	It needs to be indented more.

5.3.3 Nested Loops

Python Syntax

```
#for loop nested inside a for loop
for [first iterating variable] in [outer loop]:   # Outer loop
[statements specifying what to carry out]       # Optional
for [second iterating variable] in [inner loop]:   # Inner loop
[statements specifying what to carry out]       # Optional

#while loop nested inside a for loop
for [first iterating variable] in [outer loop]:   # Outer loop
[statements specifying what to carry out]   # Optional
while (test condition) in [inner loop]:   # Inner loop
[statements specifying what to carry out]      # Optional

while loop nested inside a while loop
while first condition:   # Outer loop
[statements specifying what to carry out]  # Optional
while the second condition:   # Inner loop
[statements specifying what to carry out]    # Optional
```

5.4 Jump Statements

- There may be times during the execution of the code when a pause or termination of the code is necessary (also known as an abnormal loop termination) due to the occurrence of a recursion statement.
- Generally, to stop or pause the looping statements, control statements are used.
- Code performance deviates from its usual pattern due to these iteration loop control statements (directed by the code).
- To instantaneously exit the body of a loop or recheck the condition(s) from the inside of the loop, these control statements are used.

5.4.1 break Statement

Syntax of break statement: The following is a syntactical representation of the break statement:

To terminate a loop, use the keyword "break".

The loop starts with the optional declared variable, which takes one element from the iterable object at a time to execute the statements in the next block until it encounters a break.

The flow of iterations through a for loop that contains a break statement is as follows (also it is shown in Fig. 5.9):

1. Loop begins
2. The for loop starts with the optional declared variable, which takes one element from the iterable object at a time to execute the statements in the next block.

3. Execute the first statement.

4. Execute the second statement.

5. Bumps into the break statement.

6. Jumps out of the loop.

7. Terminates the code execution.

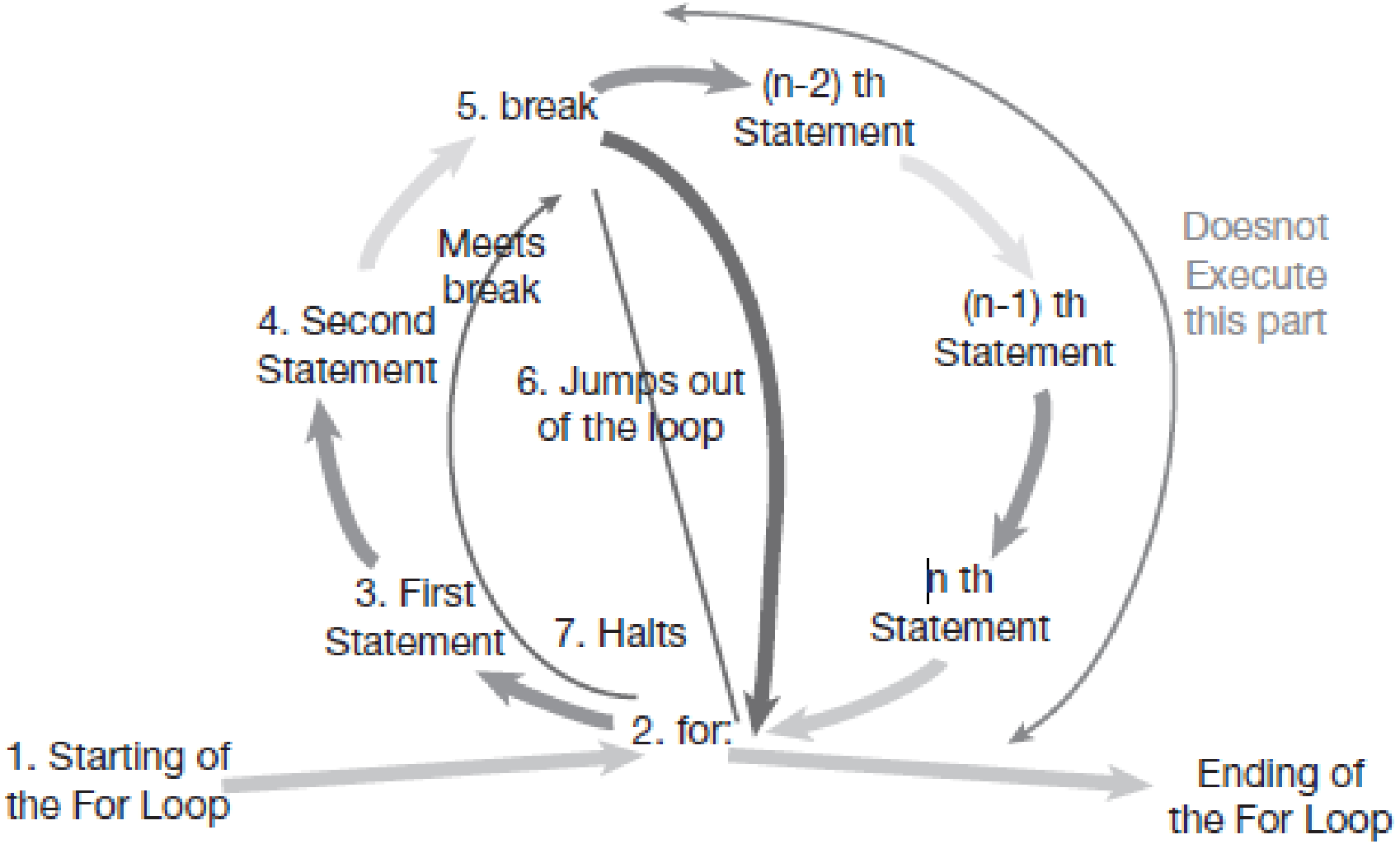
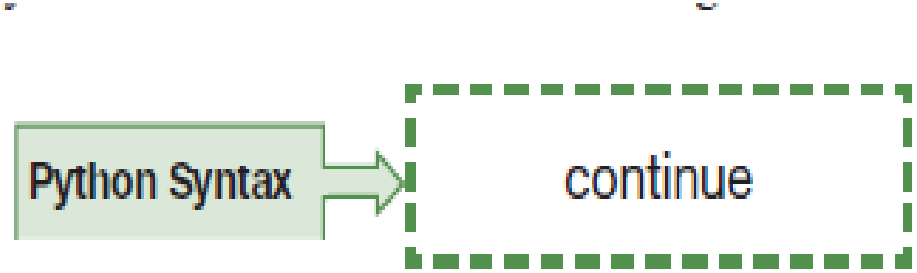


Fig. 5.9: Exemplification of the Process Flow of Break Statement in Python.

5.4.2 continue Statement

Syntax of continue: The syntax for the continue statement is given below:



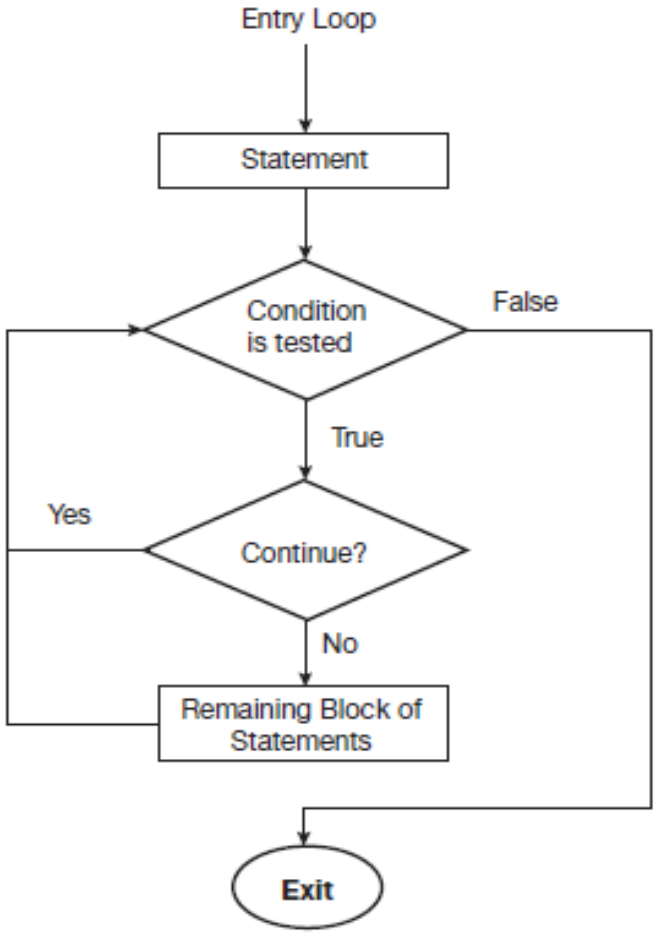


Fig. 5.12: A Flow Chart Depicting the Process of Using continue in Python Loops.

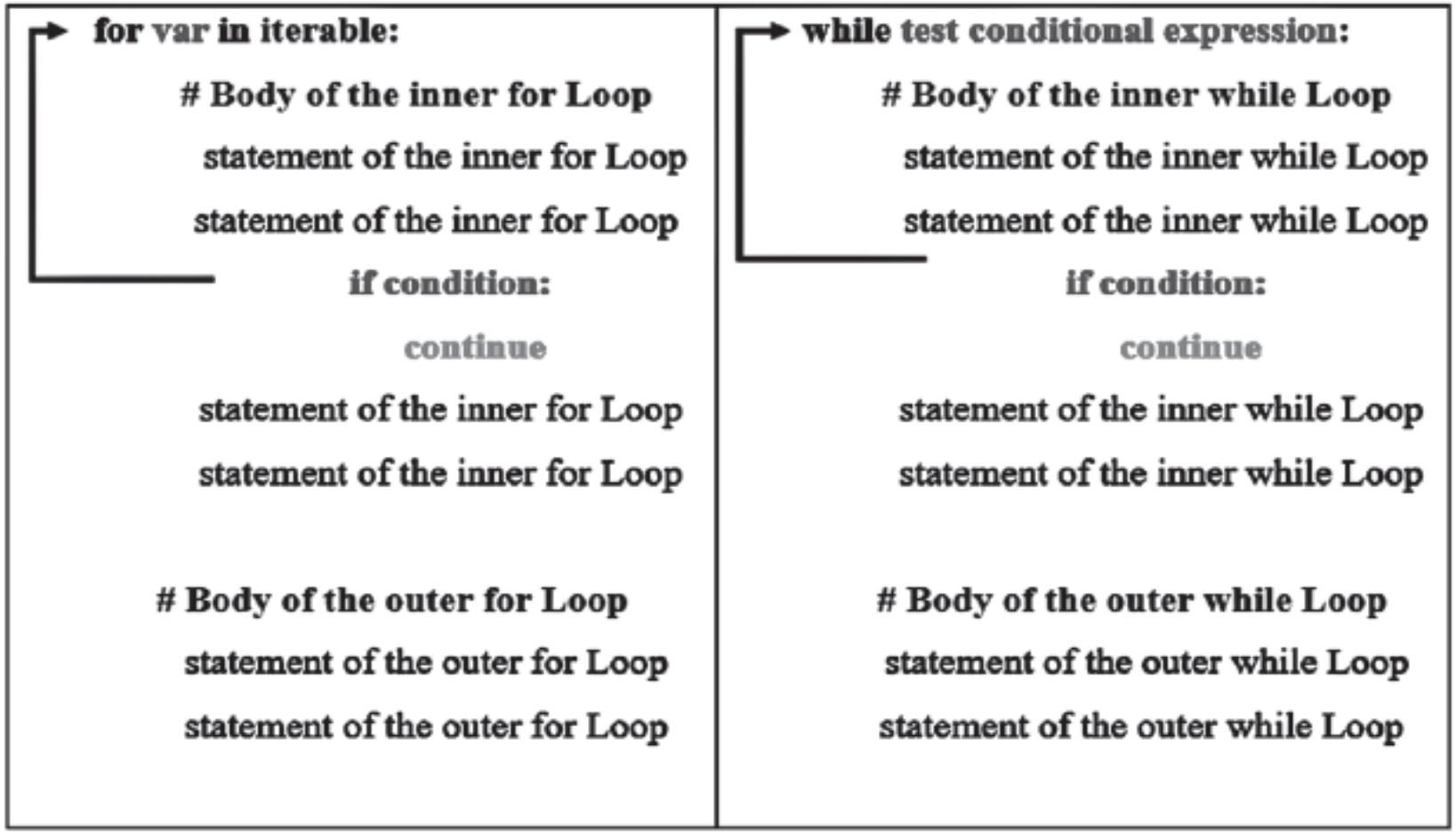


Fig. 5.13: Python Loops Flow Control With the Continue.

5.4.3 pass Statement

Need for a pass Statement:

- used to write and implement classes, empty loops, empty control statements, and functions in the future.
- helpful in some cases where the parent class will have a function that is not useful to that class. But it might be a valuable function to its child class so it can evade any fault in the base class.
- the pass statement comprehends itself as a *placeholder* in the compound statement.

- The placeholder will either be blank, or nothing will be written there. It is used in places that are supposed to include a *function with no execution* and to declare the function in a code that will be used in the future.
- body of the function, cannot be left empty as it will raise an error. The error is raised and hurled either as syntactically or systematically incorrect.
- Python pass statement is used as more details are needed to write the function's body.

Table 5.3: Differences Between the pass and continue

Based On	pass	continue
Nature of Statement	A null statement	Not a null statement
Placeholder	It can be used as a placeholder for future code.	It cannot be used as a placeholder for future code.
Ensuing Development Of Action	A Pass statement informs the interpreter regarding the non-availability of code.	The Continue statement jumps to the top line of the loop.

Based On	pass	continue
Forces Into	A Pass statement then passes the implementation to the following statement, or it tries to construe the next line of code.	The Continue statement will force the loop to start the next iteration and complete it till the end.
Action Performed	It does not perform or stay doing nothing.	Acts by jumping to the subsequent iteration
Obligatory	Mandatory when syntactically needed but practically not	Required when we want to skip the execution of remote inning statement(s) inside the loop for the current iteration
Nature of the Keyword	Pass keyword is a "no-operation" keyword	Continue keyword is used to resume a loop at the control point