

PYTHON PROGRAMMING FOR CIVIL ENGINEERS

UNIT 2: CONTROL STRUCTURES AND FLOW OF EXECUTION

LECTURE 1: BOOLEAN VALUES AND BOOLEAN OPERATORS

1. Learning Objectives

After completing this lecture, students should be able to:

- Explain what a Boolean value is and why it is important in programming.
- Identify the two Boolean values: True and False.
- Understand Boolean expressions and relational comparisons.
- Use the Boolean operators not, and, and or.
- Construct and interpret truth tables.
- Apply Boolean logic to simple Civil Engineering decision problems.
- Distinguish between a condition that produces a Boolean result and a statement that uses that result.

2. Concept Introduction

In Civil Engineering, many decisions are based on whether a condition is satisfied or not. For example: Is the measured concrete strength greater than the specified strength? Is rainfall above a warning threshold? Is the observed settlement within the permissible limit? Such questions have only two logical outcomes: Yes/No, Satisfied/Not satisfied, or True/False.

Python represents these two logical outcomes using Boolean values. Boolean logic forms the foundation of conditional statements and loops, which will be studied in the next lectures.

3. Definition: Boolean Values

A Boolean value is a logical value that can have only one of two states: True or False. In Python, True and False are Boolean constants and belong to the bool data type.

```
result = True
condition = False

print(result)
print(type(result))
```

Output:

```
True
<class 'bool'>
```

4. Why Boolean Values Are Needed

A computer must convert engineering rules and decision criteria into precise logical tests. Boolean values provide the answer to those tests.

Engineering question	Logical condition	Possible result
Does concrete strength meet the requirement?	strength >= required_strength	True / False
Is rainfall above the warning threshold?	rainfall > threshold	True / False
Is settlement within the permissible limit?	settlement <= limit	True / False
Is soil moisture greater than the target value?	moisture > target	True / False

5. Boolean Expressions

A Boolean expression is an expression that evaluates to either True or False. Comparison (relational) operators are commonly used to create Boolean expressions.

5.1 Comparison Operators

Operator	Meaning	Example
==	Equal to	a == b
!=	Not equal to	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Example 1: Simple Boolean comparisons

Python program:

```
strength = 32
required_strength = 30

print(strength >= required_strength)
print(strength < required_strength)
print(strength == required_strength)
```

Step-by-step / logical explanation:

- `strength >= required_strength` checks whether 32 is at least 30.
- The first comparison is True.
- `strength < required_strength` is False.
- `strength == required_strength` is False because the two values are not equal.

Sample output:

```
True
False
False
```

Engineering application:

Python Programming for Civil Engineers | Unit 2 | Lecture 1: Boolean Values and Boolean Operators

This is the basic logic used to check whether a measured concrete compressive strength satisfies a specified minimum strength.

6. Boolean Operators

Boolean operators combine or modify Boolean expressions. Python provides three main Boolean operators: not, and, and or.

Operator	Purpose	General idea	Example
not	Negation	Reverses True/False	not A
and	Logical conjunction	True only when both conditions are True	A and B
or	Logical disjunction	True when at least one condition is True	A or B

7. The `not` Operator

Concept introduction: `not` reverses a Boolean value. True becomes False, and False becomes True.

Definition: `not` is the logical negation operator.

Why needed: Engineering rules often require testing the opposite of a condition.

Syntax:

```
not condition
```

Explanation: Python first evaluates the condition and then reverses its Boolean result.

Example 2: Using `not`

Python program:

```
within_limit = True
print(within_limit)
print(not within_limit)
```

Step-by-step / logical explanation:

- `within\_limit` stores True.
- `not within\_limit` reverses True to False.
- Therefore, the second print statement reports that the condition is not false; rather, it is the opposite of the original value.

Sample output:

```
True
False
```

Engineering application:

If a structural check records `within\_limit = True`, then `not within\_limit` represents the condition that the value is outside the permitted limit.

8. The `and` Operator

Concept introduction: `and` is used when two or more conditions must be satisfied simultaneously.

Definition: `A and B` is True only when both A and B are True.

Why needed: Many engineering acceptance criteria require multiple conditions to be satisfied together.

Syntax:

```
condition1 and condition2
```

8.1 Truth Table for `and`

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Example 3: Concrete strength and workability criteria

Python program:

```
strength = 32
minimum_strength = 30

slump = 80
minimum_slump = 50
maximum_slump = 100

strength_ok = strength >= minimum_strength
slump_ok = minimum_slump <= slump <= maximum_slump

print(strength_ok)
print(slump_ok)
print(strength_ok and slump_ok)
```

Step-by-step / logical explanation:

- The concrete strength condition checks whether strength is at least the specified minimum.
- The slump condition checks whether slump lies between the specified limits.
- `and` combines the two Boolean results.
- The final result is True only if both requirements are satisfied.

Sample output:

```
True
True
True
```

Engineering application:

This illustrates a common quality-control decision: a concrete sample may be accepted only when both strength and workability criteria are satisfied.

9. The `or` Operator

Concept introduction: `or` is used when any one of two or more conditions is sufficient.

Definition: `A or B` is True when at least one of A or B is True.

Why needed: Some engineering warning systems are activated when any one of several threshold conditions is exceeded.

Syntax:

```
condition1 or condition2
```

9.1 Truth Table for `or`

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Example 4: Rainfall warning condition

Python program:

```
rainfall = 145
warning_limit = 100
critical_limit = 200

warning = rainfall > warning_limit
critical = rainfall > critical_limit

print(warning or critical)
```

Step-by-step / logical explanation:

- The first condition checks whether rainfall exceeds the warning limit.
- The second checks whether rainfall exceeds the critical limit.
- At 145 mm, the warning condition is True and the critical condition is False.
- Because `or` needs at least one True condition, the final result is True.

Sample output:

```
True
```

Engineering application:

The Boolean result can later be used by an `if` statement to trigger a warning message or additional analysis.

10. Combining Comparison and Boolean Operators

In practical engineering programs, comparison operators and Boolean operators are frequently combined. The comparison operators first produce Boolean values; then `and`, `or`, or `not` combines or modifies them.

```
rainfall = 175
threshold = 150
```

```
condition = rainfall > threshold
print(condition)

combined = (rainfall > 150) and (rainfall < 200)
print(combined)
```

Output:

```
True
True
```

Parentheses are recommended when combining conditions because they make the intended logic easier for beginners to read and reduce ambiguity.

11. Operator Precedence and Parentheses

When an expression contains several operators, Python follows rules of precedence. For beginner-level Boolean expressions, the safest practice is to use parentheses to show the intended engineering logic explicitly.

A useful simplified order to remember is:

1. Parentheses `()`
2. `not`
3. Comparison operators such as `>`, `<`, `>=`, `<=`, `==`, `!=`
4. `and`
5. `or`

```
A = True
B = False
C = True

print((A and C) or B)
print(A and (C or B))
```

Output:

```
True
True
```

12. Important Python Rules for Boolean Values

- `True` and `False` must begin with capital letters.
- `true` and `false` are not Python Boolean constants.
- The Boolean type is written as `bool`.
- A comparison such as `10 > 5` produces a Boolean result.
- `=` means assignment; `==` means comparison for equality.
- Use parentheses to make complex engineering conditions easy to understand.
- Indentation becomes especially important when Boolean expressions are used inside `if` statements; conditional statements will be covered in Lecture 2.

13. Basic Example → Intermediate Example → Civil Engineering Application

Basic Example: Is a value positive?

Python program:

```
value = 25
print(value > 0)
```

Step-by-step / logical explanation:

- Python compares 25 with 0.
- The comparison is True.
- The Boolean result is printed.

Sample output:

```
True
```

Intermediate Example: Checking a range

Python program:

```
temperature = 38
lower = 20
upper = 40

within_range = (temperature >= lower) and (temperature <= upper)
print(within_range)
```

Step-by-step / logical explanation:

- The first comparison checks the lower limit.
- The second comparison checks the upper limit.
- `and` requires both comparisons to be True.
- Therefore, the temperature is within the specified range.

Sample output:

```
True
```

Civil Engineering Application: Concrete acceptance screening

Python program:

```
strength = 31.5
required_strength = 30.0

slump = 90
min_slump = 50
max_slump = 100

strength_ok = strength >= required_strength
slump_ok = (slump >= min_slump) and (slump <= max_slump)

acceptable = strength_ok and slump_ok
```

```
print("Strength criterion:", strength_ok)
print("Slump criterion:", slump_ok)
print("Overall screening result:", acceptable)
```

Step-by-step / logical explanation:

- `strength\_ok` becomes True because 31.5 MPa is at least 30 MPa.
- `slump\_ok` becomes True because 90 mm lies between 50 mm and 100 mm.
- `acceptable` is the logical AND of the two criteria.
- This is only a screening example. Actual concrete acceptance must follow the applicable code, specification, sampling plan, and project requirements.

Sample output:

```
Strength criterion: True
Slump criterion: True
Overall screening result: True
```

Engineering application:

The program demonstrates how Boolean logic can convert measured field/laboratory observations into machine-readable quality-control conditions.

14. Flow of Execution: Boolean Expressions

A Boolean expression is evaluated from the input values toward a final logical result.

```
Engineering data
|
v
Evaluate comparison
(e.g., strength >= 30)
|
v
Boolean result
(True / False)
|
v
Combine with and / or
or reverse using not
|
v
Final Boolean result
```

At this stage, the program may simply print the result. In the next lecture, the Boolean result will be used to decide which block of Python statements should execute.

15. Common Mistakes Made by Beginners

Beginner mistake	Correction
Writing `true` or `false`	Python uses `True` and `False`.
Using `=` instead of `==`	`=` assigns a value; `==` compares two values.
Forgetting parentheses in complex conditions	Parentheses make the intended logic explicit and

	readable.
Confusing `and` with `or`	`and` requires all combined conditions to be True; `or` requires at least one.
Assuming `not` means 'not equal'	`not` reverses a Boolean result; `!=` is the not-equal comparison operator.
Ignoring units in engineering data	Python does not understand engineering units automatically. The programmer must use consistent units.
Using the wrong threshold	Check whether the engineering criterion is a minimum, maximum, or range before writing the condition.

16. Important Indentation Note

Boolean values by themselves do not require indentation. However, when a Boolean condition is used with an `if`, `for`, or `while` statement, Python uses indentation to identify the statements belonging to that block.

```

strength = 32
required = 30

# Boolean expression
strength_ok = strength >= required

# Used later with a conditional statement
if strength_ok:
    print("Requirement satisfied")

```

The `if` statement is introduced here only to show where Boolean logic leads. Its complete structure will be taught in Lecture 2.

17. Engineering Problem-Solving with Boolean Logic

When translating a Civil Engineering rule into Python, use the following sequence:

- Identify the engineering quantity and its unit.
- Identify the required limit, threshold, or range.
- Decide whether the rule is a minimum, maximum, equality, inequality, or combination of conditions.
- Write the comparison expression.
- Check whether the expression gives True or False for a known test value.
- If multiple requirements exist, combine them using `and` or `or`.
- Use the final Boolean result in a decision statement, which will be covered in the next lecture.

Examples of suitable Boolean logic:

Engineering problem	Boolean logic	Meaning
Concrete strength must be at least 25 MPa	<code>strength >= 25</code>	Requirement satisfied
Rainfall warning above 100 mm	<code>rainfall > 100</code>	Warning condition active
Settlement must not exceed 20 mm	<code>settlement <= 20</code>	Within limit
Temperature must lie from 20°C to 40°C	<code>20 <= temp <= 40</code>	Within range

Either of two warning criteria is exceeded	criterion1 or criterion2	At least one warning condition active
Both quality criteria must be satisfied	criterion1 and criterion2	Both requirements satisfied

18. Summary of the Lecture

- Boolean values represent logical states: True and False.
- Boolean expressions commonly arise from comparisons.
- The comparison operators are `==`, `!=`, `>`, `<`, `>=`, and `<=`.
- `not` reverses a Boolean result.
- `and` is True only when all combined conditions are True.
- `or` is True when at least one combined condition is True.
- Boolean logic is essential for converting engineering rules into programmable decision criteria.
- Boolean expressions become especially powerful when used with conditional statements and loops.

19. Quick Reference

Construct	Example	Typical engineering use
Boolean value	True / False	Store a logical status
Comparison	strength >= 30	Check a requirement
not	not within_limit	Reverse a logical condition
and	strength_ok and slump_ok	Require multiple criteria
or	rainfall_warning or flood_warning	Trigger when any criterion occurs

20. Practice Questions

A. Short-Answer Questions

13. What is a Boolean value in Python?
14. Name the two Boolean constants in Python.
15. What is the data type of True and False?
16. What is the difference between `=` and `==`?
17. What does the `not` operator do?
18. When does `A and B` become True?
19. When does `A or B` become False?
20. Write the six comparison operators used in Python.
21. Why are parentheses useful in Boolean expressions?
22. Give one Civil Engineering example where a Boolean condition is useful.

B. Conceptual Questions

23. Why are Boolean values important for engineering decision-making?
24. Explain the difference between `and` and `or` using a construction quality-control example.
25. Why is `not` different from the `!=` operator?
26. Construct the truth table for `and`.
27. Construct the truth table for `or`.
28. Why should engineering units be considered when writing Boolean conditions?

29. How can a range condition be represented using Boolean logic?
30. What is the role of Boolean expressions in conditional statements?
31. Why is it safer for beginners to use parentheses in complex logical expressions?
32. Explain how an engineering specification can be translated into a Boolean expression.

C. Python Programming Exercises

33. Write a program to check whether a given number is positive.
34. Write a program to check whether a given number is equal to zero.
35. Write a program to check whether a temperature is within a specified range.
36. Write a program to check whether a measured length is greater than a minimum required length.
37. Write a program using ``not`` to determine whether a measurement is outside a permissible limit.
38. Write a program using ``and`` to check two independent requirements.
39. Write a program using ``or`` to check whether at least one of two warning conditions is active.
40. Write a program to check whether a rainfall value is between two specified limits.
41. Write a program to check whether both concrete strength and slump satisfy given limits.
42. Write a program that accepts two Boolean values and prints the results of ``and``, ``or``, and ``not``.

D. Civil Engineering-Based Programming Problems

43. Concrete quality check: Read compressive strength and a specified minimum strength. Produce a Boolean result indicating whether the strength criterion is satisfied.
44. Rainfall threshold: Read daily rainfall and a warning threshold. Produce True if rainfall exceeds the threshold.
45. Settlement check: Read observed settlement and permissible settlement. Produce True if the observed settlement is within the limit.
46. Survey observation check: Read an observed distance and its permissible range. Determine whether the observation lies within the acceptable range.
47. Construction quality screening: Read concrete strength and slump. Use Boolean operators to determine whether both specified criteria are satisfied.

E. Viva Questions with Answers

Question	Answer
What is the Boolean data type in Python?	The <code>`bool`</code> data type, which represents True or False.
Is <code>`True`</code> the same as <code>`true`</code> in Python?	No. Python Boolean constants are written as <code>`True`</code> and <code>`False`</code> with capital first letters.
What is the result of <code>`10 > 5`</code> ?	True.
What is the result of <code>`True and False`</code> ?	False.
What is the result of <code>`True or False`</code> ?	True.

21. Faculty Note: Transition to Lecture 2

Students should understand that Boolean logic answers the question 'Is this condition True or False?' The next step is to make Python act differently depending on that answer. This is the purpose of conditional statements: ``if``, ``if-else``, and ``if-elif-else``.

PYTHON PROGRAMMING FOR CIVIL ENGINEERS

UNIT 2: CONTROL STRUCTURES AND FLOW OF EXECUTION

LECTURE 2: CONDITIONAL STATEMENTS, LOOPS AND NESTED LOOPS

1. Learning Objectives

- Understand how Python changes its flow of execution according to a condition.
- Use if, if-else and if-elif-else statements correctly.
- Understand the importance of indentation in Python blocks.
- Choose between for and while loops based on the engineering problem.
- Use loops to repeat calculations for stations, samples, loads, activities and observations.
- Understand nested loops and trace their execution.
- Translate simple Civil Engineering problem statements into algorithms and Python programs.

2. Prerequisite: Boolean Logic

Conditional statements and loops depend on Boolean expressions. A Boolean expression produces True or False.

```
strength = 32
required_strength = 30
strength_ok = strength >= required_strength
print(strength_ok)
```

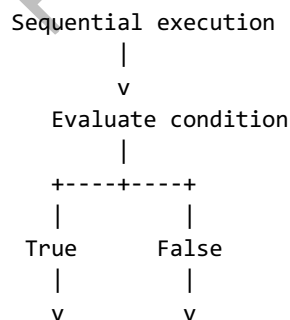
Output:

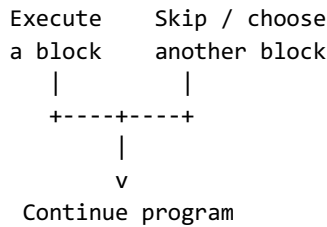
True

In this lecture, the Boolean result is used to control which statements Python executes and how many times statements are repeated.

3. Flow of Execution

Normally, Python executes statements from top to bottom. Control structures change this normal sequential flow.





Loops create repetition: after executing a loop body, Python may return to the loop condition or proceed to the next item.

4. Conditional Statement: if

4.1 Concept Introduction

An if statement allows Python to execute a block only when a specified condition is True.

4.2 Definition

The if statement is a one-way decision structure. If the condition is True, its indented block executes; if False, that block is skipped.

4.3 Why is it needed?

Engineering programs frequently need to act only when a criterion is satisfied—for example, displaying a warning when rainfall exceeds a threshold.

4.4 Syntax

```

if condition:
    statement_1
    statement_2
  
```

Explanation of syntax:

- `if` is the Python keyword.
- `condition` is a Boolean expression.
- A colon `:` marks the beginning of the block.
- Indented statements belong to the if block.
- If the condition is False, Python skips the entire indented block.

4.5 Basic Example: Check a value

Python program:

```

value = 25

if value > 0:
    print("The value is positive")
  
```

Step-by-step / logical explanation:

- Python evaluates `value > 0`.
- The result is True.
- Python enters the indented block.

- The message is printed.

Sample output:

```
The value is positive
```

4.6 Civil Engineering Example: Rainfall warning

Python program:

```
rainfall = 125
warning_limit = 100

if rainfall > warning_limit:
    print("Rainfall warning: threshold exceeded")
```

Step-by-step / logical explanation:

- Rainfall is 125 mm.
- The warning threshold is 100 mm.
- $125 > 100$ is True.
- The warning message is therefore executed.

Sample output:

```
Rainfall warning: threshold exceeded
```

Engineering application:

Such a condition can be part of a larger rainfall-monitoring or flood early-warning workflow. Actual operational thresholds must come from the applicable hydrological study or authority.

5. Conditional Statement: if-else

5.1 Concept Introduction

An if-else statement provides two alternative paths: one for True and another for False.

5.2 Definition

The if block executes when the condition is True; otherwise the else block executes.

5.3 Why is it needed?

Many engineering decisions have exactly two outcomes, such as Satisfactory/Not Satisfactory or Safe for the specified criterion/Does not satisfy the specified criterion.

5.4 Syntax

```
if condition:
    statement_if_true
else:
    statement_if_false
```

The two blocks must have matching indentation, and `else` must align with `if`.

5.5 Basic Example: Even or odd

Python program:

```
number = 18

if number % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

Step-by-step / logical explanation:

- The remainder operator `%` finds the remainder after division by 2.
- If the remainder is zero, the number is even.
- Otherwise the else block executes.

Sample output:

```
Even number
```

5.6 Civil Engineering Example: Concrete strength criterion

Python program:

```
strength = 28.5
required_strength = 30.0

if strength >= required_strength:
    print("Strength criterion satisfied")
else:
    print("Strength criterion not satisfied")
```

Step-by-step / logical explanation:

- The measured strength is 28.5 MPa.
- The specified minimum used in this example is 30 MPa.
- $28.5 \geq 30$ is False.
- Therefore the else block executes.

Sample output:

```
Strength criterion not satisfied
```

Engineering application:

The program demonstrates the logic of a minimum-strength check. Actual concrete acceptance should follow the relevant design code, specification and statistical acceptance procedure rather than a single simplified comparison.

6. if vs if-else

Feature	if	if-else
Number of main paths	One conditional action	Two alternative actions
When condition is True	if block executes	if block executes

When condition is False	Nothing in the if block executes	else block executes
Typical use	Warning, optional action	Pass/fail type decision

7. Conditional Statement: if-elif-else

7.1 Concept Introduction

When an engineering quantity can fall into more than two categories, if-elif-else provides multiple mutually exclusive choices.

7.2 Definition

Python checks conditions from top to bottom. The first condition that is True has its block executed; the remaining elif/else blocks are skipped.

7.3 Why is it needed?

Classification problems often have multiple ranges—for example, classifying a rainfall event as low, moderate, high or very high according to chosen study thresholds.

7.4 Syntax

```
if condition_1:
    statement_1
elif condition_2:
    statement_2
elif condition_3:
    statement_3
else:
    statement_default
```

`elif` means 'else if'. Any number of `elif` branches can be used, but the conditions should be arranged carefully.

7.5 Basic Example: Grade classification

Python program:

```
marks = 72

if marks >= 75:
    grade = "Distinction"
elif marks >= 60:
    grade = "First Class"
elif marks >= 50:
    grade = "Second Class"
else:
    grade = "Below Second Class"

print(grade)
```

Step-by-step / logical explanation:

- Python first checks marks >= 75; it is False.
- It checks marks >= 60; it is True.
- Python executes that block and does not test the later alternatives.

- The result is First Class.

Sample output:

First Class

7.6 Civil Engineering Example: Rainfall classification

Python program:

```
rainfall = 165

if rainfall < 50:
    category = "Low"
elif rainfall < 100:
    category = "Moderate"
elif rainfall < 200:
    category = "High"
else:
    category = "Very High"

print("Rainfall category:", category)
```

Step-by-step / logical explanation:

- 165 is not less than 50.
- 165 is not less than 100.
- 165 is less than 200, so the High category is selected.
- The remaining branch is skipped.

Sample output:

Rainfall category: High

Engineering application:

These thresholds are illustrative only. In an actual hydrological application, rainfall categories should be defined from the relevant basin, station, duration and authority/study.

8. if-elif-else vs Multiple Independent if Statements

Feature	if-elif-else	Multiple independent if
Testing	Stops after first True branch	Every if condition is evaluated
Use when	Only one category/action should be selected	Several conditions may all need action
Classification	Usually appropriate	Can produce multiple actions
Example	Low/Moderate/High rainfall class	Check several independent warning flags

Example of the distinction:

```
x = 75

if x >= 50:
    print("At least 50")
```

```
elif x >= 70:  
    print("At least 70")
```

Only the first message is printed because the first condition is already True. If the requirements are independent, separate `if` statements may be more appropriate.

9. Nested Conditional Statements

A conditional statement placed inside another conditional statement is called a nested conditional. Nesting should be used only when the second decision logically depends on the first.

```
strength = 32  
slump = 90  
  
if strength >= 30:  
    if 50 <= slump <= 100:  
        print("Both criteria satisfied")  
    else:  
        print("Strength is adequate, but slump is outside the range")  
else:  
    print("Strength criterion not satisfied")
```

Output:

```
Both criteria satisfied
```

The inner test is performed only after the outer condition is True.

10. Looping Constructs

A loop repeats a block of statements. Repetition is essential in engineering because calculations are often performed for many observations, stations, samples, elements, load cases or construction activities.

10.1 for Loop

Concept introduction

A for loop is generally used when Python can iterate through a known collection or sequence, or when a known number of repetitions is required.

Definition

The for loop takes items one at a time from an iterable such as a list or a range and executes the loop body for each item.

Why is it needed?

It avoids writing the same calculation repeatedly for every station, sample, activity or case.

Syntax

```
for variable in sequence:  
    statement_1  
    statement_2
```

The loop variable receives one item at a time. The indented block executes once for each item.

10.2 Basic Example: Repeat for five values

Python program:

```
for i in range(1, 6):  
    print(i)
```

Step-by-step / logical explanation:

- `range(1, 6)` generates 1, 2, 3, 4 and 5.
- The loop variable `i` takes each value in sequence.
- The print statement executes five times.
- The upper limit 6 is excluded.

Sample output:

```
1  
2  
3  
4  
5
```

10.3 Intermediate Example: Calculate total quantity

Python program:

```
quantities = [12, 15, 10, 18]  
total = 0  
  
for q in quantities:  
    total = total + q  
  
print("Total quantity:", total)
```

Step-by-step / logical explanation:

- The list contains four quantities.
- `total` starts at zero.
- The loop adds each quantity to total.
- After all four values are processed, total is 55.

Sample output:

```
Total quantity: 55
```

Engineering application:

This pattern can be used for quantities of concrete, excavation, reinforcement or other work items when the data are already available.

10.4 Civil Engineering Example: Total concrete for activities

Python program:

```
activities = ["Footing", "Column", "Beam", "Slab"]
volumes = [18.5, 12.0, 15.5, 30.0]

total_volume = 0

for volume in volumes:
    total_volume = total_volume + volume

print("Total concrete volume:", total_volume, "m3")
```

Step-by-step / logical explanation:

- The two lists represent activity names and corresponding concrete volumes.
- The loop processes each volume.
- Each value is accumulated in `total\_volume`.
- The final total is 76.0 m³.

Sample output:

```
Total concrete volume: 76.0 m3
```

Engineering application:

The same accumulation idea is useful in quantity estimation. In larger projects, activity data may come from spreadsheets or databases.

11. while Loop

11.1 Concept Introduction

A while loop repeats a block as long as its condition remains True.

11.2 Definition

A while loop is a condition-controlled loop. The condition is tested before every iteration.

11.3 Why is it needed?

It is useful when the number of iterations is not known in advance and repetition should continue until a target, tolerance or stopping condition is reached.

11.4 Syntax

```
while condition:
    statement_1
    statement_2
```

The loop must change something that eventually makes the condition False. Otherwise, an infinite loop can occur.

```
while condition:
    ...
    update_variable
```

11.5 Basic Example: Count from 1 to 5

Python program:

```
i = 1

while i <= 5:
    print(i)
    i = i + 1
```

Step-by-step / logical explanation:

- `i` starts at 1.
- Python checks `i <= 5`.
- The body prints i.
- The update increases i by 1.
- When i becomes 6, the condition is False and the loop stops.

Sample output:

```
1
2
3
4
5
```

11.6 Engineering Example: Iterative settlement check

Python program:

```
settlement = 5.0
limit = 20.0
increment = 3.0

while settlement <= limit:
    print("Settlement:", settlement, "mm")
    settlement = settlement + increment

print("Limit exceeded.")
```

Step-by-step / logical explanation:

- The initial settlement is 5 mm.
- The loop continues while settlement is at or below 20 mm.
- Each iteration increases the illustrative value by 3 mm.
- When the value becomes greater than the limit, the loop stops.

Sample output:

```
Settlement: 5.0 mm
Settlement: 8.0 mm
Settlement: 11.0 mm
Settlement: 14.0 mm
Settlement: 17.0 mm
Settlement: 20.0 mm
Limit exceeded.
```

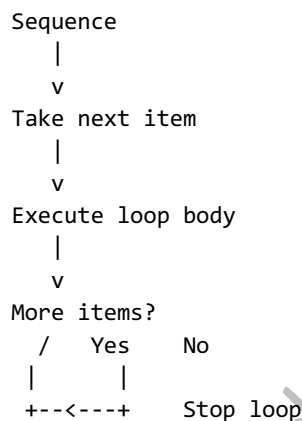
Engineering application:

This demonstrates condition-controlled repetition. The increment is only an illustrative mechanism; real settlement prediction requires an appropriate geotechnical model and input data.

12. for Loop vs while Loop

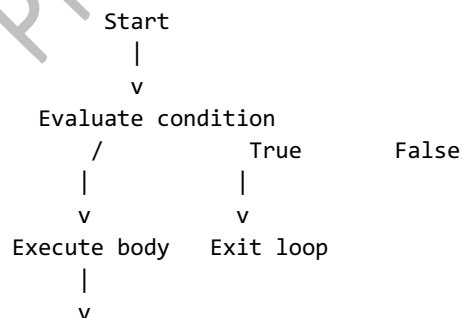
Feature	for loop	while loop
Control basis	Items in a sequence / known repetitions	Boolean condition
Typical use	Process all observations or repeat N times	Continue until target/tolerance/condition changes
Counter update	Often handled by the loop mechanism	Usually must be updated explicitly
Risk of infinite loop	Lower for simple range/collection loops	Higher if condition never becomes False
Civil example	Process 20 survey observations	Iterate until error is below tolerance

13. Flow of Execution of a for Loop



For each item, the body executes exactly once. When no items remain, Python continues with the first statement after the loop.

14. Flow of Execution of a while Loop



```
Update value
|
+-----> condition
```

The update step is critical. Without an appropriate update, the loop may never terminate.

15. Nested Loops

15.1 Concept Introduction

A nested loop is a loop placed inside another loop. For every iteration of the outer loop, the inner loop completes all of its iterations.

15.2 Definition

A nested loop consists of an outer loop and one or more inner loops. The inner loop belongs to the indented block of the outer loop.

15.3 Why is it needed?

Engineering data often have two or more dimensions—for example, several stations with several observations at each station, or several load cases containing several load components.

15.4 Syntax

```
for outer_item in outer_sequence:
    for inner_item in inner_sequence:
        statement
```

15.5 Basic Example: Multiplication table

Python program:

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i, "x", j, "=", i * j)
```

Step-by-step / logical explanation:

- Outer loop selects 1, 2 and 3.
- For each outer value, the inner loop runs through 1, 2 and 3.
- Thus the inner loop executes three times for each outer iteration.
- Total executions of the print statement = $3 \times 3 = 9$.

Sample output:

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
```

15.6 Civil Engineering Example: Stations and observations

Python program:

```
stations = ["S1", "S2", "S3"]
observations = [10, 20, 30]

for station in stations:
    print("Station:", station)
    for observation in observations:
        print("  Observation:", observation)
```

Step-by-step / logical explanation:

- The outer loop processes one station at a time.
- For each station, the inner loop processes all three observations.
- The inner loop therefore runs three times for each station.
- Total observation-processing iterations = $3 \times 3 = 9$.

Sample output:

```
Station: S1
  Observation: 10
  Observation: 20
  Observation: 30
Station: S2
  Observation: 10
  Observation: 20
  Observation: 30
Station: S3
  Observation: 10
  Observation: 20
  Observation: 30
```

Engineering application:

The structure illustrates two-level processing. In a real surveying program, each station would normally have its own associated observation data rather than reusing the same list.

15.7 Civil Engineering Example: Load cases and load components

Python program:

```
load_cases = ["LC1", "LC2", "LC3"]
components = ["Dead Load", "Live Load"]

for case in load_cases:
    print("Load case:", case)
    for component in components:
        print("  Check:", component)
```

Step-by-step / logical explanation:

- The outer loop selects a load case.
- The inner loop checks each load component for that case.
- All components are considered before moving to the next load case.

Sample output:

```
Load case: LC1
  Check: Dead Load
  Check: Live Load
Load case: LC2
  Check: Dead Load
  Check: Live Load
Load case: LC3
  Check: Dead Load
  Check: Live Load
```

Engineering application:

This is a structural-analysis data-processing pattern. Actual load combinations and code checks must follow the applicable structural design standard.

16. Nested Loop Execution: Important Point

If the outer loop runs m times and the inner loop runs n times for every outer iteration, the inner statement executes $m \times n$ times.

```
for i in range(1, 4):      # 3 outer iterations
    for j in range(1, 5):  # 4 inner iterations
        print(i, j)       # 3 x 4 = 12 executions
```

This idea is important when estimating program execution time and when designing data-processing algorithms.

17. Indentation: A Critical Python Rule

Python uses indentation to define blocks. Unlike languages that use braces, Python identifies the body of an if statement or loop by indentation.

```
strength = 32

if strength >= 30:
    print("Criterion satisfied")
    print("Continue processing")

print("Program continues")
```

The first two print statements belong to the if block. The final print statement is outside the block.

Common indentation errors:

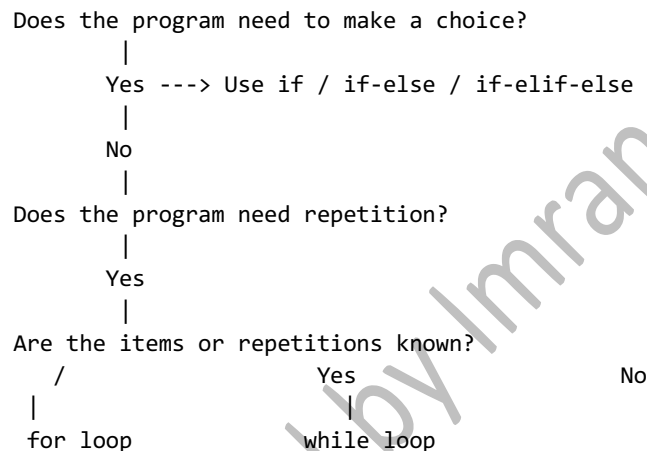
- Indenting a block inconsistently.
- Placing a statement that should be inside a loop outside the loop.
- Using `else` or `elif` at the wrong indentation level.
- Accidentally adding an extra level of indentation.
- Mixing tabs and spaces in a way that causes indentation errors.

18. Common Logical Errors

Error	Why it happens	How to avoid it
Wrong threshold	Engineering requirement misunderstood	Write the requirement in words before coding
Wrong comparison	Minimum/maximum direction reversed	Ask: 'Should value be $\geq$ or $\leq$ limit?'
Independent if used for exclusive categories	All conditions are tested	Use if-elif-else when only one category should be selected
Infinite while loop	Loop variable/condition never changes appropriately	Trace the condition and update each iteration
Off-by-one error with range	Upper bound is excluded	Remember `range(start, stop)` excludes stop
Wrong nesting	Statements indented under the wrong loop	Trace outer iteration and inner iteration separately
Unit inconsistency	Values use different units	Convert to consistent units before comparison/calculation

19. Engineering Problem-Solving: Choosing a Condition or Loop

A useful decision rule is:



In practice, a Civil Engineering problem may require both. For example, a loop may process 100 rainfall observations, while an if statement classifies each observation.

20. Integrated Engineering Applications

Problem	Logic required	Python concept	Engineering interpretation
Problem 1: Concrete strength screening	For each test result, check whether it meets a specified minimum.	for loop + if	Process a series of laboratory observations.
Problem 2: Soil classification by simple threshold rules	Assign a category based on ranges of a measured property.	if-elif-else	Classify each sample according to stated illustrative criteria.
Problem 3: Rainfall warning	Check whether rainfall exceeds a threshold.	if	Trigger a warning condition.
Problem 4: Survey	Repeat a calculation for	for loop	Process each station

observation processing	several stations.		observation.
Problem 5: Iterative engineering calculation	Repeat until an error or tolerance condition is reached.	while loop	Useful for condition-controlled numerical iteration.
Problem 6: Structural load cases	Process multiple load cases and multiple components.	nested for loops	Two-dimensional repetition.
Problem 7: Construction quantities	Calculate or accumulate quantities for several activities.	for loop	Automate repetitive quantity calculations.
Problem 8: Monitoring records	For each station, inspect several time observations and classify threshold exceedances.	nested loop + if	Two-level data processing.

21. Worked Engineering Problem: Multiple Concrete Samples

Problem statement: A laboratory has compressive-strength results for five concrete samples. Write a program to count how many samples satisfy a specified minimum strength of 30 MPa.

Logic required:

1. Store the sample strengths.
2. Set the required minimum strength.
3. Start a counter at zero.
4. Use a for loop to process every sample.
5. For each sample, use if to check whether it meets the minimum.
6. If satisfied, increase the counter.
7. Print the final count.

Python program:

```

strengths = [28.5, 31.0, 29.5, 32.5, 35.0]
required = 30.0
satisfied_count = 0

for strength in strengths:
    if strength >= required:
        satisfied_count = satisfied_count + 1

print("Samples satisfying criterion:", satisfied_count)

```

Sample output:

```
Samples satisfying criterion: 3
```

Engineering interpretation: Three of the five supplied sample values meet the simplified minimum-strength screening criterion. Actual acceptance decisions require the applicable code provisions and sampling/statistical rules.

22. Worked Engineering Problem: Survey Observation Range

Problem statement: Four measured distances are to be checked against a permissible range of 49.5 m to 50.5 m.

Algorithm/pseudocode:

8. Read/store the observations.
9. For each observation, check whether it is greater than or equal to 49.5 and less than or equal to 50.5.
10. If true, report 'Within range'.
11. Otherwise, report 'Outside range'.

```
distances = [50.1, 49.2, 50.4, 50.8]
lower = 49.5
upper = 50.5

for d in distances:
    if lower <= d <= upper:
        print(d, "m: Within range")
    else:
        print(d, "m: Outside range")
```

Sample output:

```
50.1 m: Within range
49.2 m: Outside range
50.4 m: Within range
50.8 m: Outside range
```

Engineering interpretation: The program identifies observations that satisfy the stated numerical range. Survey quality-control procedures should also consider instrument accuracy, observation method, tolerances and applicable specifications.

23. Worked Engineering Problem: Iterative Calculation

Problem statement: Start with an error of 10.0 units and reduce it by half during each iteration until it becomes less than 0.5 units. Count the iterations.

```
error = 10.0
tolerance = 0.5
iterations = 0

while error >= tolerance:
    print("Iteration:", iterations + 1, "Error:", error)
    error = error / 2
    iterations = iterations + 1

print("Final error:", error)
print("Iterations:", iterations)
```

Sample output:

```
Iteration: 1 Error: 10.0
Iteration: 2 Error: 5.0
Iteration: 3 Error: 2.5
```

Iteration: 4 Error: 1.25
Iteration: 5 Error: 0.625
Final error: 0.3125
Iterations: 5

Engineering interpretation: This illustrates the control logic of an iterative calculation. The mathematical update used here is only a teaching example; an actual engineering numerical method must define a valid convergence equation and stopping criterion.

24. Quick Comparison of the Main Constructs

Construct	Main question	Typical use
if	Is the condition True?	One-way decision
if-else	Which of two paths applies?	Two-way decision
if-elif-else	Which category/path applies?	Multiple alternatives
for	What items/repetitions must be processed?	Known collection or repetition count
while	Should repetition continue?	Condition-controlled iteration
Nested loop	Is there repetition within repetition?	Stations × observations, cases × components

25. Important Points to Remember

- Every condition must ultimately be interpretable as True or False.
- Always put a colon after `if`, `elif`, `else`, `for` and `while` headers.
- Indentation defines the body of a Python block.
- `if-elif-else` is generally suitable when alternatives are mutually exclusive.
- `for` is convenient for processing every item in a known sequence.
- `while` is suitable when continuation depends on a condition.
- A while loop needs a correctly designed update or stopping mechanism.
- In nested loops, the inner loop completes for each outer-loop iteration.
- Do not choose a loop simply because repetition exists; first identify what controls the repetition.
- Use meaningful variable names such as `rainfall`, `strength`, `station`, `load\_case`, and `settlement`.

26. Practice Section

A. Short-Answer Questions

12. What is a conditional statement?
13. What is the purpose of the colon after an `if` condition?
14. What is the role of indentation in Python?
15. What happens when an `if` condition is False and there is no `else`?
16. What is the purpose of `elif`?
17. What is a loop?
18. What is the difference between a sequence-controlled and condition-controlled loop?
19. What does `range(1, 5)` generate?
20. What is an infinite loop?
21. What is a nested loop?

B. Conceptual Questions

22. Explain why Boolean expressions are the foundation of conditional statements.
23. Compare if and if-else using a concrete-strength example.
24. Why is if-elif-else generally appropriate for classifying observations into exclusive categories?
25. Why might separate if statements be preferable when several warning conditions can occur simultaneously?
26. Explain the difference between for and while loops using Civil Engineering examples.
27. Why is updating the loop-control variable important in a while loop?
28. Explain how a nested loop processes station and observation data.
29. What is an off-by-one error in a for loop using range?
30. Why should engineering units be made consistent before applying conditions?
31. Explain how a loop and a condition can be used together to process engineering observations.

C. Python Programming Exercises

32. Write a program using if to check whether a number is greater than 100.
33. Write an if-else program to determine whether a measured length is within a specified maximum limit.
34. Write an if-elif-else program to classify a temperature into three ranges.
35. Write a for loop to print numbers from 1 to 10.
36. Write a for loop to calculate the total of five measured values.
37. Write a while loop to print values from 1 to 10.
38. Write a while loop that continues until an error becomes less than a specified tolerance.
39. Write a nested loop to print a 4×4 grid of values.
40. Write a program that uses a loop and if statement to count values greater than a threshold.
41. Write a program using if-elif-else inside a loop to classify a list of observations.

D. Civil Engineering-Based Programming Problems

42. Concrete quality: Given compressive strengths of several samples, count and report how many meet a specified minimum strength.
43. Rainfall analysis: Given rainfall observations for several days, count how many exceed a warning threshold and display each warning day.
44. Surveying: Given a list of observed distances, identify which observations fall within a permissible tolerance range.
45. Construction quantities: Given quantities for multiple activities, calculate the total quantity using a for loop.
46. Structural analysis: Given several load cases, use nested loops to process multiple load components for each case.

E. Viva Questions with Answers

Question	Answer
What is the main purpose of an if statement?	To execute a block only when its condition is True.
What is the difference between if and if-else?	if provides a conditional path; if-else provides alternative True and False paths.
When should a for loop be preferred?	When processing a known sequence or a known number of repetitions.
When is a while loop useful?	When repetition should continue while a condition remains True, especially when the number of iterations is not known beforehand.
What is a nested loop?	A loop placed inside another loop; the inner loop executes for each iteration of the outer loop.

27. Faculty Teaching Notes

Suggested classroom sequence:

47. Begin with a real engineering decision: 'Does the measured concrete strength satisfy the specified minimum?'
48. Ask students to identify the Boolean condition.
49. Demonstrate if, then add else for the opposite result.
50. Introduce a third category and show why elif is useful.
51. Ask: 'What if this calculation must be performed for 50 samples?' Introduce for.
52. Ask: 'What if we do not know how many iterations are needed?' Introduce while.
53. Demonstrate nested loops using stations × observations or load cases × components.
54. Trace one complete example on the board before students run the code.

28. Lecture Summary

Conditional statements allow a Python program to make decisions, while loops allow it to repeat operations. `if` handles a one-way decision, `if-else` handles two alternatives, and `if-elif-else` handles multiple alternatives. `for` is convenient for known collections or repetitions, while `while` is appropriate for condition-controlled repetition. Nested loops provide a structured way to process two-dimensional or multi-level engineering data. Together, these constructs form the core of engineering problem-solving programs in Python.

PYTHON PROGRAMMING FOR CIVIL ENGINEERS

UNIT 2: CONTROL STRUCTURES AND FLOW OF EXECUTION

LECTURE 3: CONTROL STATEMENTS — `break`, `continue`, `pass`

1. Learning Objectives

- Explain the purpose of control statements in Python loops.
- Use `break` to terminate a loop when further iterations are unnecessary.
- Use `continue` to skip selected iterations while keeping the loop running.
- Use `pass` as a placeholder when a valid block is required but no action is currently needed.
- Distinguish clearly between `break`, `continue`, and `pass`.
- Apply these statements to Civil Engineering data-processing problems.
- Trace loop execution and identify common indentation and logical errors.

2. Prerequisite Knowledge

Students should already understand Boolean values, comparison operators, `if`, `if-else`, `for`, and `while` loops. Control statements modify or deliberately leave unchanged the normal flow of execution.

Normal loop:

Start → Check condition / next item → Execute body → Next iteration → Repeat

|
+→ `break`: exit loop
+→ `continue`: next iteration
+→ `pass`: no operation; continue normally

3. What Are Control Statements?

Control statements are statements that alter the normal sequence of execution. In this lecture, three Python statements are studied.

Statement	Basic action	Effect on loop
<code>break</code>	Terminate the loop	Loop ends immediately
<code>continue</code>	Skip current iteration	Loop proceeds to next iteration
<code>pass</code>	Do nothing	Normal execution continues

4. Why Are They Needed in Engineering Programs?

Engineering data sets can contain critical events, invalid observations, or modules that are not yet implemented. Control statements allow a program to respond appropriately.

Situation	Statement	Reason
Critical event means no more scanning is required	<code>break</code>	Stop processing

One invalid/missing record should be ignored	continue	Process later records
A future module is not yet implemented	pass	Maintain valid program structure
A numerical iteration has reached its stopping criterion	break	Terminate iteration

5. `break` Statement

5.1 Concept Introduction

The `break` statement immediately terminates the nearest enclosing loop.

5.2 Definition

When Python executes `break`, it leaves the current loop and continues with the first statement after that loop.

5.3 Why is it needed?

Use `break` when continuing the loop is no longer necessary—for example, after detecting the first critical threshold exceedance.

5.4 Syntax

```
for item in sequence:
    if condition:
        break
```

The same idea can be used inside a `while` loop.

5.5 Basic Example: Stop when a value is found

Python program:

```
numbers = [10, 20, 30, 40, 50]

for number in numbers:
    print(number)
    if number == 30:
        break

print("Loop ended")
```

Step-by-step / logical explanation:

- The loop starts with 10.
- 10 and 20 are printed.
- At 30, the condition is True.
- `break` exits the loop.
- 40 and 50 are not processed.
- The statement after the loop executes.

Sample output:

```
10
20
```

30

Loop ended

5.6 Intermediate Example: Search for a target

Python program:

```
observations = [12.5, 15.2, 18.4, 21.0, 24.6]
target = 21.0

for value in observations:
    if value == target:
        print("Target observation found")
        break
    print("Checking:", value)
```

Step-by-step / logical explanation:

- Values are checked sequentially.
- The first three values do not match.
- At 21.0 the condition becomes True.
- `break` terminates the search.
- 24.6 is not checked.

Sample output:

```
Checking: 12.5
Checking: 15.2
Checking: 18.4
Target observation found
```

5.7 Civil Engineering Application: First critical rainfall observation

Python program:

```
rainfall = [42, 65, 88, 105, 175, 120]
critical_limit = 150

for value in rainfall:
    print("Rainfall:", value, "mm")
    if value > critical_limit:
        print("Critical threshold exceeded")
        break

print("Monitoring scan completed")
```

Step-by-step / logical explanation:

- Rainfall is examined in sequence.
- Values at or below 150 mm do not trigger `break`.
- At 175 mm the condition is True.
- `break` stops this scan immediately.
- The later 120 mm is not processed in this scan.

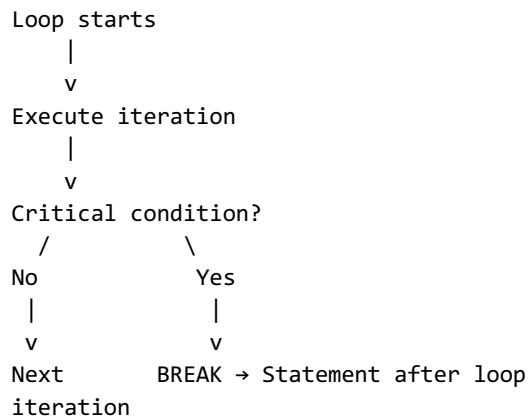
Sample output:

```
Rainfall: 42 mm
Rainfall: 65 mm
Rainfall: 88 mm
Rainfall: 105 mm
Rainfall: 175 mm
Critical threshold exceeded
Monitoring scan completed
```

Engineering application:

Useful when the immediate objective is to detect the first critical event. If every record must be analyzed, do not stop the loop at the first event.

5.8 Flowchart for `break`



5.9 Important Points

- `break` terminates only the nearest enclosing loop.
- It does not terminate the entire Python program.
- Statements after the loop can still execute.
- In nested loops, `break` normally exits only the inner loop where it appears.
- Use it when no further iterations of that loop are required.

6. `continue` Statement

6.1 Concept Introduction

The `continue` statement skips the remainder of the current iteration and proceeds to the next iteration.

6.2 Definition

When Python executes `continue`, statements later in the current loop body are skipped for that iteration.

6.3 Why is it needed?

It is useful when a particular engineering record is invalid, missing, or irrelevant, but the remaining records still need to be processed.

6.4 Syntax

```
for item in sequence:
    if condition:
        continue
    process(item)
```

6.5 Basic Example: Skip one value

Python program:

```
numbers = [10, 20, 30, 40, 50]

for number in numbers:
    if number == 30:
        continue
    print(number)
```

Step-by-step / logical explanation:

- The loop reaches 30.
- The condition becomes True.
- `continue` skips the print statement for that iteration.
- The loop then processes 40 and 50.

Sample output:

```
10
20
40
50
```

6.6 Intermediate Example: Skip negative measurements

Python program:

```
values = [12, -5, 18, 0, 24]

for value in values:
    if value < 0:
        continue
    print("Processing:", value)
```

Step-by-step / logical explanation:

- Negative values are identified.
- `continue` skips only those values.
- Zero and positive values are processed.
- The loop continues through the complete list.

Sample output:

```
Processing: 12
Processing: 18
Processing: 0
Processing: 24
```

6.7 Civil Engineering Application: Skip invalid survey observations

Python program:

```
distances = [50.2, 49.8, -1, 50.1, 49.9]

for distance in distances:
    if distance < 0:
        print("Invalid observation skipped")
        continue

    print("Valid distance:", distance, "m")
```

Step-by-step / logical explanation:

- Each distance is examined.
- The negative value is treated as invalid for this simple example.
- `continue` skips normal processing for that record.
- All later valid observations are still processed.

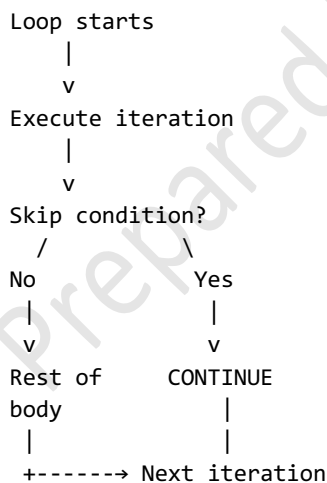
Sample output:

```
Valid distance: 50.2 m
Valid distance: 49.8 m
Invalid observation skipped
Valid distance: 50.1 m
Valid distance: 49.9 m
```

Engineering application:

Actual survey validation must consider instrument accuracy, field procedure, tolerance and applicable specifications; the negative-value test is only a programming illustration.

6.8 Flowchart for `continue`



6.9 Important Points

- `continue` does not terminate the loop.
- It affects only the current iteration.
- Statements after `continue` are skipped in that iteration.

- The next iteration begins.
- In a `while` loop, update the control variable appropriately before reaching `continue`.

6.10 Common `while`-Loop Error with `continue`

Consider this problematic pattern:

```
i = 1
while i <= 5:
    if i == 3:
        continue
    print(i)
    i = i + 1
```

When `i` becomes 3, the program reaches `continue` before increasing `i`. The value remains 3 forever, producing an infinite loop.

A safer version is:

```
i = 1
while i <= 5:
    if i == 3:
        i = i + 1
        continue
    print(i)
    i = i + 1
```

Output:

```
1
2
4
5
```

7. `pass` Statement

7.1 Concept Introduction

Unlike `break` and `continue`, `pass` does not control loop termination or iteration selection. It simply performs no operation.

7.2 Definition

`pass` is a null statement used where Python requires a statement but the programmer intentionally has no action to perform yet.

7.3 Why is it needed?

Python requires an indented statement inside blocks such as `if`, loops, functions and classes. `pass` provides a valid placeholder.

7.4 Syntax

```
if condition:
    pass
```

7.5 Basic Example: Placeholder

Python program:

```
value = 25

if value > 0:
    pass

print("Program continues")
```

Step-by-step / logical explanation:

- The condition is True.
- `pass` performs no operation.
- Python continues with the next statement.

Sample output:

```
Program continues
```

7.6 Intermediate Example: Future module

Python program:

```
def process_survey_data():
    pass

print("Function structure created")
```

Step-by-step / logical explanation:

- The function has a valid body because `pass` is present.
- No survey processing occurs yet.
- The placeholder can later be replaced by the required implementation.

Sample output:

```
Function structure created
```

Engineering application:

Useful during staged development of engineering software, where the overall program structure is prepared before every module is implemented.

7.7 Civil Engineering Application: Reserved processing category

Python program:

```
soil_type = "Organic"

if soil_type == "Rock":
    print("Process rock data")
elif soil_type == "Soil":
    print("Process soil data")
else:
    pass
```

```
print("Data screening completed")
```

Step-by-step / logical explanation:

- The value matches neither of the implemented categories.
- The `else` block executes `pass`.
- No action is taken for that category.
- Execution continues normally.

Sample output:

```
Data screening completed
```

Engineering application:

`pass` can temporarily represent a category whose processing rules are not yet implemented. It should not be mistaken for an engineering calculation.

7.8 Important Points

- `pass` means 'do nothing'.
- It does not stop the loop.
- It does not skip the current iteration.
- It is mainly a placeholder.
- A real engineering requirement should eventually be implemented and validated.

8. `break` vs `continue` vs `pass`

Feature	break	continue	pass
Main purpose	Terminate loop	Skip current iteration	Do nothing
Loop continues?	No	Yes	Yes
Current iteration	Ends loop	Ends current iteration	Continues normally
Typical use	Critical event / target found	Invalid or irrelevant record	Placeholder
Effect on next iteration	None	Starts next iteration	Normal flow
Common engineering example	Stop at first critical threshold	Skip invalid observation	Reserve future module

9. Visual Comparison of Execution

break:

Iteration 1 → Iteration 2 → BREAK → Exit loop

continue:

Iteration 1 → Iteration 2 → CONTINUE → Iteration 3 → ...

pass:

Iteration 1 → Iteration 2 → PASS → remaining statements → Iteration 3

10. Control Statements in Nested Loops

In nested loops, `break` and `continue` normally affect the nearest enclosing loop in which they appear.

10.1 Nested loop with `break`

Python program:

```
for station in range(1, 4):
    print("Station", station)

    for observation in range(1, 5):
        if observation == 3:
            break
        print("  Observation", observation)
```

Step-by-step / logical explanation:

- The outer loop selects each station.
- The inner loop processes observations.
- When observation equals 3, the inner loop breaks.
- The outer loop then proceeds to the next station.

Sample output:

```
Station 1
  Observation 1
  Observation 2
Station 2
  Observation 1
  Observation 2
Station 3
  Observation 1
  Observation 2
```

10.2 Nested loop with `continue`

Python program:

```
for station in range(1, 3):
    print("Station", station)

    for observation in range(1, 4):
        if observation == 2:
            continue
        print("  Observation", observation)
```

Step-by-step / logical explanation:

- For each station, the inner loop considers observations 1, 2 and 3.
- Observation 2 is skipped.
- Observation 3 is still processed.
- The outer loop is unaffected.

Sample output:

Station 1
Observation 1
Observation 3
Station 2
Observation 1
Observation 3

11. Engineering Problem-Solving: Choosing a Control Statement

Do I want to stop the current loop?

|
Yes → break

Do I want to ignore only this iteration?

|
Yes → continue

Do I need a valid block but no action yet?

|
Yes → pass

Ask yourself: 'What should happen to the current iteration and what should happen to later iterations?'

Question	Choose
No more iterations of this loop are required	break
Only this record should be ignored	continue
The block is intentionally empty for now	pass

12. Engineering Applications

Problem	Logic required	Python concept	Engineering use
First critical rainfall event	Stop at first threshold exceedance	break	Early event detection
Invalid survey observations	Skip selected records	continue	Data validation
Concrete sample screening	Stop/continue depending on objective	break or continue	Laboratory data processing
Iterative calculation	Stop when criterion is reached	break	Numerical iteration
Multiple load cases	Skip irrelevant components	continue	Structural data processing
Future analysis module	Leave block temporarily empty	pass	Program development

13. Worked Engineering Problem: Multiple Concrete Samples

Problem statement: Given compressive-strength results, count the samples that satisfy a minimum strength of 30 MPa. Invalid negative values are skipped.

1. Store the strength observations.
2. Set the minimum required strength.
3. For each observation, skip negative values.

4. For valid values, check the minimum-strength criterion.
5. Increase the counter when the criterion is satisfied.
6. Print the final count.

```
strengths = [28.5, 31.0, -1, 29.5, 32.5, 35.0]
required = 30.0
satisfied_count = 0

for strength in strengths:
    if strength < 0:
        continue

    if strength >= required:
        satisfied_count = satisfied_count + 1

print("Samples satisfying criterion:", satisfied_count)
```

Sample output:

```
Samples satisfying criterion: 3
```

Engineering interpretation: Three valid samples satisfy the simplified minimum-strength screening condition. Actual concrete acceptance requires the applicable code, specification and statistical acceptance procedure.

14. Worked Engineering Problem: Survey Observations

Problem statement: Given measured distances, skip invalid negative entries and calculate the total of valid observations.

```
distances = [50.2, -1, 49.8, 50.1, -2]

total = 0
count = 0

for d in distances:
    if d < 0:
        continue

    total = total + d
    count = count + 1

print("Valid observations:", count)
print("Total valid distance:", total, "m")
```

Sample output:

```
Valid observations: 3
Total valid distance: 150.1 m
```

Engineering interpretation: `continue` prevents invalid entries from affecting the accumulation while allowing the complete data set to be scanned.

15. Worked Engineering Problem: Iterative Stopping Condition

Problem statement: Reduce an illustrative error by half each iteration until it is below 0.5. Stop when the tolerance is achieved.

```
error = 10.0
tolerance = 0.5
iterations = 0

while True:
    iterations = iterations + 1
    error = error / 2
    print("Iteration:", iterations, "Error:", error)

    if error < tolerance:
        break

print("Converged")
```

Sample output:

```
Iteration: 1 Error: 5.0
Iteration: 2 Error: 2.5
Iteration: 3 Error: 1.25
Iteration: 4 Error: 0.625
Iteration: 5 Error: 0.3125
Converged
```

Engineering interpretation: This illustrates event-based termination using `break`. The numerical update is only a teaching example; actual numerical methods require a valid governing equation and convergence criterion.

16. Common Mistakes Made by Beginners

Mistake	Problem	Correction
Using `break` to skip one bad record	Entire loop stops	Use `continue`
Using `continue` to terminate the loop	Later iterations still run	Use `break`
Thinking `pass` skips an iteration	Remaining body statements still execute	Use `continue`
Using `continue` before updating a while counter	Can cause an infinite loop	Update control variable appropriately
Assuming `break` exits all nested loops	Usually only nearest loop exits	Use an appropriate outer-loop design/flag if needed
Using `pass` as a finished calculation	No calculation occurs	Implement and validate the required method
Incorrect indentation	Wrong block or IndentationError	Indent consistently
Wrong threshold direction	Incorrect engineering decision	Write the rule in words first
Ignoring units	Invalid comparison/calculation	Use consistent engineering units

17. Important Syntax and Rules

- Use lowercase keywords: `break`, `continue`, `pass`.

- ``break`` and ``continue`` are normally used inside loops.
- ``pass`` can be used in loops, conditionals, functions and classes.
- Only the nearest enclosing loop is affected by ``break`` or ``continue``.
- ``pass`` has no computational effect.
- Place control statements at the correct indentation level.
- Test small examples and trace each iteration before applying the logic to large data sets.

18. Quick Reference with Civil Engineering Examples

Construct	Question answered	Example
<code>break</code>	Should this loop stop now?	Stop after first critical rainfall
<code>continue</code>	Should this observation be skipped?	Ignore invalid survey reading
<code>pass</code>	Is there intentionally no action yet?	Future soil-processing module
<code>break + if</code>	Has a stopping condition occurred?	if error < tolerance: break
<code>continue + if</code>	Is this record unsuitable?	if value < 0: continue

19. Practice Section

A. Short-Answer Questions

7. What is a control statement?
8. What is the purpose of ``break``?
9. What is the purpose of ``continue``?
10. What is the purpose of ``pass``?
11. Does ``break`` terminate the entire Python program?
12. Does ``continue`` terminate the loop?
13. Does ``pass`` skip the current iteration?
14. Which statement is suitable for skipping an invalid observation?
15. Which statement is suitable for stopping at a critical threshold?
16. Why can ``continue`` cause an infinite loop in a poorly written ``while`` loop?

B. Conceptual Questions

17. Explain ``break`` and ``continue`` using a rainfall example.
18. Why is ``pass`` called a null statement?
19. Why should ``pass`` not be considered an engineering calculation?
20. How does ``break`` behave in a nested loop?
21. How does ``continue`` behave in a nested loop?
22. Why is updating a while-loop control variable important when using ``continue``?
23. Give a survey-processing situation where ``continue`` is preferable to ``break``.
24. Give an engineering search situation where ``break`` is preferable to ``continue``.
25. How do control statements help when processing large engineering data sets?
26. Explain how ``if``, ``continue``, and ``break`` can be combined in one engineering program.

C. Python Programming Exercises

27. Use ``break`` to stop printing numbers when the value reaches 7.
28. Use ``continue`` to skip even numbers from a list.
29. Use ``pass`` inside an ``if`` block.
30. Search a list for a target and stop using ``break`` when it is found.
31. Skip negative values using ``continue``.
32. Write a while loop using ``break`` when a counter reaches 10.
33. Write a safe while loop using ``continue`` so the counter always increases.
34. Write a nested loop where ``break`` terminates only the inner loop.
35. Write a nested loop where ``continue`` skips one observation for each station.
36. Combine ``continue`` for invalid data and ``break`` for a critical threshold.

D. Civil Engineering-Based Programming Problems

37. Concrete testing: Given compressive strengths, stop processing when the first value exceeds a specified critical value.
38. Surveying: Given measured distances, skip invalid negative entries and calculate the total of valid observations.
39. Rainfall monitoring: Skip invalid values and stop when a critical rainfall threshold is detected.
40. Structural load cases: Use nested loops and stop checking a case when a specified critical condition is found.
41. Construction quantity processing: Skip zero/invalid quantities and accumulate the remaining quantities.

E. Viva Questions with Answers

Question	Answer
What does <code>`break`</code> do?	It immediately exits the nearest enclosing loop.
What does <code>`continue`</code> do?	It skips the rest of the current iteration and proceeds to the next iteration.
What does <code>`pass`</code> do?	It performs no operation and acts as a placeholder.
Which statement skips an invalid record without stopping the loop?	<code>`continue`</code> .
Which statement stops a loop when no further iterations are required?	<code>`break`</code> .

20. Faculty Teaching Notes

42. Start with a normal loop using five simple numbers.
43. Add ``break`` and ask students to predict which values will be printed.
44. Replace ``break`` with ``continue`` and compare the result.
45. Replace the statement with ``pass`` and demonstrate that normal execution continues.
46. Use a survey-observation list containing one invalid value.
47. Demonstrate the dangerous placement of ``continue`` in a ``while`` loop before the counter update.
48. Finish with a nested station × observation example and identify which loop is affected.

Teaching emphasis: Ask students after every control statement, 'What happens to the current iteration?' and 'What happens to the next iteration?'

21. Lecture Summary

``break`` terminates the nearest loop; ``continue`` skips only the current iteration; and ``pass`` performs no operation. These statements are especially useful in Civil Engineering programs for threshold detection, data validation, iterative calculations, nested data processing and staged software development. Correct indentation, clear stopping conditions and consistent engineering units are essential for reliable programs.

Prepared by Imran, Asst.Prof, MCET

PYTHON PROGRAMMING FOR CIVIL ENGINEERS

UNIT 2: CONTROL STRUCTURES AND FLOW OF EXECUTION

LECTURE 4: USE OF LOOPS AND CONDITIONS IN ENGINEERING PROBLEM SOLVING

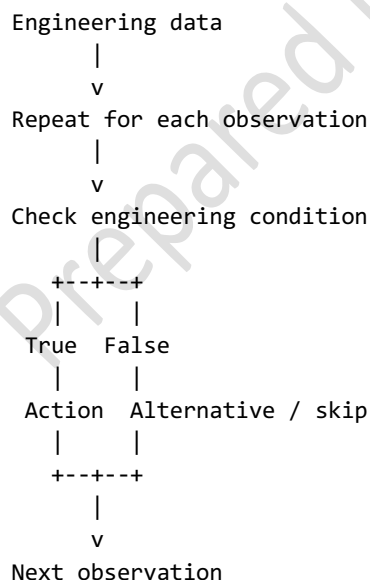
1. Learning Objectives

- Recognize repetitive and decision-based patterns in Civil Engineering problems.
- Translate an engineering problem into input, processing logic, conditions, repetition and output.
- Decide when to use `if`, `if-else`, `if-elif-else`, `for`, `while`, nested loops, `break` and `continue`.
- Develop simple algorithms and pseudocode before writing Python code.
- Use loops and conditions together to process engineering observations.
- Build beginner-friendly programs for concrete testing, soil classification, rainfall, surveying, structural loads and construction quantities.
- Interpret Python results from an engineering perspective and recognize the limitations of simplified examples.

2. Concept Introduction

Civil Engineering calculations rarely involve only one value. A field engineer may have observations from several survey stations, a laboratory may have many material samples, a hydrologist may analyze rainfall for many days, and a structural engineer may evaluate several load cases. At the same time, engineering specifications contain decision rules such as minimum strength, maximum settlement, permissible error or threshold rainfall.

Loops provide repetition; conditions provide decisions. Combining them allows Python to process engineering data systematically.



3. The Core Idea: Condition + Repetition

A condition answers: 'What should happen if a rule is satisfied?' A loop answers: 'How many times or for which observations should this process be repeated?'

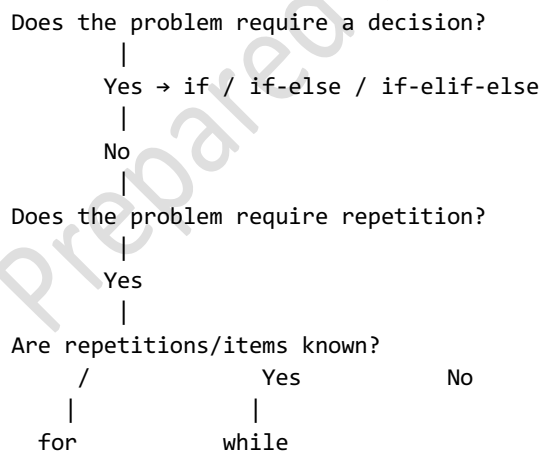
Requirement in a problem	Python concept
Make one decision	`if`
Choose between two alternatives	`if-else`
Choose among several categories	`if-elif-else`
Process a known set of observations	`for`
Repeat until a condition changes	`while`
Process data at two levels	Nested loops
Stop when a critical event occurs	`break`
Ignore one unsuitable record	`continue`

4. Engineering Problem-Solving Workflow

A reliable beginner workflow is:

1. Read the engineering problem carefully.
2. Identify the given data and units.
3. Identify the required output.
4. Write the engineering rule in ordinary language.
5. Determine whether a decision, repetition, or both are required.
6. Choose the appropriate Python control structure.
7. Write a short algorithm/pseudocode.
8. Translate the algorithm into Python.
9. Test the program with simple known values.
10. Interpret the result using engineering terminology.
11. Check units, limits, assumptions and applicable standards.

5. How to Decide Whether to Use a Condition or a Loop



Often:

loop + condition = process many engineering observations and make a decision for each one

Important: A real engineering problem may need both a loop and a condition. The choice is based on the problem logic, not on the subject area alone.

6. General Pattern: Loop + Condition

The most common engineering data-processing pattern is:

```
for value in data:
    if condition:
        action
```

This means: take each value, test the engineering rule, and perform an action only when the rule is satisfied.

6.1 Basic Example: Count values above a limit

Python program:

```
values = [10, 25, 40, 15, 60]
limit = 30
count = 0

for value in values:
    if value > limit:
        count = count + 1

print("Values above limit:", count)
```

Step-by-step / logical explanation:

- The list contains five values.
- The `for` loop examines every value.
- The `if` condition checks whether each value is greater than 30.
- The counter increases only for values above 30.
- The values 40 and 60 satisfy the condition, so the count is 2.

Sample output:

```
Values above limit: 2
```

7. Application 1: Concrete Compressive Strength Checking

7.1 Problem statement

A set of concrete compressive-strength test results is available. Write a program to determine how many results satisfy a specified minimum strength.

7.2 Logic required

12. Store the test results.
13. Store the specified minimum strength.
14. Set a counter to zero.
15. Process each test result using a `for` loop.
16. Use `if` to compare strength with the minimum.

17. Increase the counter if the criterion is satisfied.
18. Display the count.

7.3 Pseudocode

```
START
Read/store strength values
Set required strength
Set count = 0
FOR each strength:
    IF strength >= required:
        increase count
Display count
STOP
```

7.4 Python program

Python program:

```
strengths = [28.5, 31.0, 29.5, 32.5, 35.0]
required_strength = 30.0
satisfied = 0

for strength in strengths:
    if strength >= required_strength:
        satisfied = satisfied + 1

print("Required strength:", required_strength, "MPa")
print("Samples satisfying criterion:", satisfied)
```

Step-by-step / logical explanation:

- The loop processes five samples.
- The condition uses `>=` because the stated criterion is a minimum.
- Three values—31.0, 32.5 and 35.0—satisfy the criterion.
- The counter records the number of satisfying samples.

Sample output:

```
Required strength: 30.0 MPa
Samples satisfying criterion: 3
```

Engineering application:

This is a simplified screening example. Actual concrete acceptance depends on the relevant concrete code, grade, sampling procedure, characteristic strength requirements and statistical acceptance criteria.

7.5 Extending the program: report each sample

```
strengths = [28.5, 31.0, 29.5, 32.5, 35.0]
required_strength = 30.0

for i in range(len(strengths)):
    if strengths[i] >= required_strength:
        print("Sample", i + 1, ":", strengths[i], "MPa - Satisfies")
    else:
        print("Sample", i + 1, ":", strengths[i], "MPa - Does not satisfy")
```

This combines a loop with `if-else` so that every observation receives a classification.

8. Application 2: Soil Classification Using Thresholds

8.1 Problem statement

Suppose a teaching exercise requires classification of soil samples into three categories based on a selected numerical property. The thresholds below are illustrative and are not a substitute for a formal soil-classification standard.

8.2 Logic required

If property < 20	→ Category A
Else if property < 40	→ Category B
Else	→ Category C

Because there are three mutually exclusive ranges, `if-elif-else` is appropriate.

8.3 Python program

Python program:

```
samples = [15, 27, 42, 35, 18]

for value in samples:
    if value < 20:
        category = "Category A"
    elif value < 40:
        category = "Category B"
    else:
        category = "Category C"

    print(value, "->", category)
```

Step-by-step / logical explanation:

- The loop takes one sample value at a time.
- Python tests the first range.
- If it is false, the second range is tested.
- If both are false, the else category is selected.
- Only one category is assigned to each sample.

Sample output:

```
15 -> Category A
27 -> Category B
42 -> Category C
35 -> Category B
18 -> Category A
```

Engineering application:

Actual soil classification should follow the relevant geotechnical classification system and required laboratory parameters. The example demonstrates programming logic only.

9. Application 3: Rainfall Threshold Analysis

9.1 Problem statement

Given daily rainfall observations, identify the days on which rainfall exceeds a warning threshold.

9.2 Logic required

19. Store rainfall observations.
20. Set warning threshold.
21. Loop through each day's observation.
22. Use `if` to test whether rainfall exceeds the threshold.
23. Report the observation when the threshold is exceeded.

9.3 Python program

Python program:

```
rainfall = [45, 82, 110, 65, 155, 92, 180]
warning_limit = 100

for day, value in enumerate(rainfall, start=1):
    if value > warning_limit:
        print("Day", day, ":", value, "mm - Warning")
```

Step-by-step / logical explanation:

- `enumerate` provides a simple day number along with each rainfall value.
- The condition checks whether rainfall is above 100 mm.
- Only days 3, 5 and 7 satisfy the condition.

Sample output:

```
Day 3 : 110 mm - Warning
Day 5 : 155 mm - Warning
Day 7 : 180 mm - Warning
```

Engineering application:

Thresholds must be defined for the appropriate rainfall duration and purpose. A single daily threshold is not automatically a flood prediction rule.

9.4 Counting threshold exceedances

```
rainfall = [45, 82, 110, 65, 155, 92, 180]
warning_limit = 100
warning_days = 0

for value in rainfall:
    if value > warning_limit:
        warning_days = warning_days + 1

print("Number of warning observations:", warning_days)
```

Output:

```
Number of warning observations: 3
```

10. Application 4: Surveying Observations and Tolerance Checking

10.1 Problem statement

A target distance is 50.00 m and the permissible tolerance is ± 0.05 m. Check several observations and report whether each lies within the tolerance.

10.2 Engineering logic

```
Lower limit = target - tolerance
Upper limit = target + tolerance
Accept if lower limit <= observation <= upper limit
```

10.3 Python program

Python program:

```
observations = [50.02, 49.93, 50.08, 49.97]
target = 50.00
tolerance = 0.05

lower = target - tolerance
upper = target + tolerance

for obs in observations:
    if lower <= obs <= upper:
        print(obs, "m - Within tolerance")
    else:
        print(obs, "m - Outside tolerance")
```

Step-by-step / logical explanation:

- The acceptable range is 49.95 m to 50.05 m.
- Each observation is tested against both limits.
- Observations 50.02 and 49.97 are within the range.
- 49.93 and 50.08 are outside the range.

Sample output:

```
50.02 m - Within tolerance
49.93 m - Outside tolerance
50.08 m - Outside tolerance
49.97 m - Within tolerance
```

Engineering application:

The tolerance used is illustrative. Actual survey tolerances depend on survey class, instrument, method, network design and project requirements.

11. Application 5: Construction Quantity Calculation

11.1 Problem statement

Calculate total concrete quantity for several construction activities and identify activities exceeding a planning threshold.

11.2 Python program

Python program:

```
activities = ["Footing", "Column", "Beam", "Slab"]
volumes = [18.5, 12.0, 15.5, 30.0]
large_quantity = 20.0
total = 0

for i in range(len(activities)):
    total = total + volumes[i]

    if volumes[i] > large_quantity:
        print(activities[i], "has a quantity above", large_quantity, "m3")

print("Total concrete:", total, "m3")
```

Step-by-step / logical explanation:

- The loop processes each activity and its corresponding quantity.
- Each quantity is added to the total.
- The condition identifies quantities greater than 20 m³.
- The slab quantity of 30 m³ exceeds the planning threshold.
- The total quantity is 76 m³.

Sample output:

```
Slab has a quantity above 20.0 m3
Total concrete: 76.0 m3
```

Engineering application:

The calculation demonstrates repetitive quantity processing. Project quantity take-off should be based on approved drawings, measurement rules and the applicable method of measurement.

12. Application 6: Structural Load Case Processing

12.1 Problem statement

For a teaching example, process several load cases and check whether the total of listed load components exceeds an illustrative screening limit.

12.2 Python program

Python program:

```
load_cases = {
    "LC1": [20, 15],
    "LC2": [30, 25],
    "LC3": [15, 10]
}

limit = 50

for case in load_cases:
```

```

total = 0

for load in load_cases[case]:
    total = total + load

print(case, "Total load =", total, "kN")

if total > limit:
    print(" Screening limit exceeded")

```

Step-by-step / logical explanation:

- The outer loop processes one load case.
- The inner loop adds its load components.
- The condition checks the resulting total against 50 kN.
- LC2 totals 55 kN and exceeds the illustrative screening limit.

Sample output:

```

LC1 Total load = 35 kN
LC2 Total load = 55 kN
  Screening limit exceeded
LC3 Total load = 25 kN

```

Engineering application:

Real structural design uses specified load combinations, factored actions, load paths and code provisions. This example illustrates nested loops and conditions rather than a complete structural design procedure.

13. Application 7: Processing Multiple Stations and Observations

13.1 Problem statement

Suppose each surveying station has several observations. Identify observations outside an illustrative acceptable range.

13.2 Python program

Python program:

```

station_data = {
    "S1": [49.98, 50.02, 50.08],
    "S2": [50.01, 49.97, 50.03],
    "S3": [49.90, 50.00, 50.04]
}

lower = 49.95
upper = 50.05

for station in station_data:
    print("Station:", station)

    for observation in station_data[station]:
        if lower <= observation <= upper:
            print(" ", observation, "- OK")

```

```
else:
    print(" ", observation, "- Check")
```

Step-by-step / logical explanation:

- The outer loop selects a station.
- The inner loop processes all observations belonging to that station.
- Each observation is checked against the permissible range.
- Values outside the range are flagged for further review.

Sample output:

```
Station: S1
  49.98 - OK
  50.02 - OK
  50.08 - Check
Station: S2
  50.01 - OK
  49.97 - OK
  50.03 - OK
Station: S3
  49.9 - Check
  50.0 - OK
  50.04 - OK
```

Engineering application:

This is a two-level data-processing structure. Actual survey adjustment and quality assessment require the appropriate observation model and error analysis.

14. Application 8: Iterative Engineering Problem Using `while`

14.1 Problem statement

An iterative calculation starts with an error of 8.0 and reduces the error by half at each iteration. Stop when the error becomes less than 0.5.

14.2 Logic

24. Initialize error and tolerance.
25. Check whether error is still above or equal to tolerance.
26. Perform one iteration.
27. Update the error.
28. Repeat until the stopping condition becomes false.

14.3 Python program

Python program:

```
error = 8.0
tolerance = 0.5
iteration = 0

while error >= tolerance:
    iteration = iteration + 1
```

```

    error = error / 2
    print("Iteration:", iteration, "Error:", error)

print("Final error:", error)

```

Step-by-step / logical explanation:

- The `while` loop continues while error is at least 0.5.
- Each iteration halves the error.
- The loop terminates when the updated error becomes less than the tolerance.
- The example demonstrates a condition-controlled iteration.

Sample output:

```

Iteration: 1 Error: 4.0
Iteration: 2 Error: 2.0
Iteration: 3 Error: 1.0
Iteration: 4 Error: 0.5
Iteration: 5 Error: 0.25
Final error: 0.25

```

Engineering application:

Actual numerical methods require a valid governing equation, convergence analysis and an appropriate stopping criterion. The halving rule is only a teaching illustration.

15. Using `break` in Engineering Problem Solving

Use `break` when a defined event means the current search or iteration should terminate.

15.1 Example: Stop at first critical observation

Python program:

```

values = [12, 18, 25, 31, 15]
critical = 30

for value in values:
    if value > critical:
        print("Critical value detected:", value)
        break

```

Step-by-step / logical explanation:

- Values are examined sequentially.
- At 31, the critical condition is True.
- `break` stops the loop.
- The final 15 is not processed.

Sample output:

```

Critical value detected: 31

```

16. Using `continue` in Engineering Problem Solving

Use `continue` when one observation should be excluded but the rest of the data must still be processed.

16.1 Example: Skip invalid data

Python program:

```
rainfall = [50, -1, 75, 110, -1, 80]

total = 0
count = 0

for value in rainfall:
    if value < 0:
        continue

    total = total + value
    count = count + 1

print("Valid observations:", count)
print("Total rainfall:", total, "mm")
```

Step-by-step / logical explanation:

- Negative values are treated as invalid for this simplified example.
- `continue` skips them.
- Valid observations continue to be accumulated.
- Four valid values are processed.

Sample output:

```
Valid observations: 4
Total rainfall: 315 mm
```

Engineering application:

Actual rainfall quality control requires source-specific missing-data and quality flags. A negative-value test is only an instructional example.

17. Conditions Inside Loops: A General Engineering Pattern

```
for observation in observations:
    if observation is invalid:
        continue

    if observation is critical:
        break

    if observation satisfies criterion:
        process observation
    else:
        record exception
```

This pattern is powerful but should be kept simple for beginners. Each condition should correspond to a clearly stated engineering rule.

18. How to Choose Between `for` and `while`

Engineering question	Recommended construct	Example
Process all readings in a known list	`for`	Process 100 survey observations
Repeat exactly 20 times	`for` with `range`	Generate 20 trial calculations
Continue until error < tolerance	`while`	Iterative numerical calculation
Continue until a critical event occurs	`while` or loop + `break`	Monitoring/search task
Process stations and observations	Nested `for`	Station × observation data

When the number of iterations is known or the data are already in a collection, `for` is usually easier for beginners. When continuation depends on a changing condition, `while` is often more natural.

19. Algorithm vs Python Program

An algorithm describes the solution steps independently of Python syntax. Writing the algorithm first helps prevent logical errors.

Stage	Question	Example
Problem	What is required?	Find observations outside tolerance
Input	What data are needed?	Observed distances
Rule	What engineering criterion applies?	Lower ≤ observation ≤ upper
Repetition	How many observations?	Every observation
Decision	What should happen?	Report OK / Check
Output	What should be displayed?	Status for each observation

20. Common Engineering Logic Patterns

Pattern	Typical Python structure	Civil Engineering example
Threshold check	`if value > limit`	Rainfall warning
Minimum requirement	`if value >= minimum`	Concrete strength
Maximum permissible value	`if value <= maximum`	Settlement limit
Range check	`if lower <= value <= upper`	Survey tolerance
Classification	`if-elif-else`	Soil/property categories
Count qualifying values	Loop + counter + `if`	Samples meeting strength
Accumulate quantities	Loop + accumulator	Concrete quantity
Skip invalid values	Loop + `continue`	Missing/invalid observation
Stop at critical event	Loop + `break`	First critical threshold
Two-dimensional processing	Nested loops	Stations × observations

21. Common Mistakes in Engineering Programs

Mistake	Example problem	Prevention
Wrong inequality	Using `>` for a minimum requirement	Translate the rule into words first
Wrong units	Comparing mm with m	Convert to consistent units
Off-by-one error	Wrong number of `range` iterations	Remember the stop value is excluded
Infinite while loop	Control variable never changes	Trace the update and stopping

		condition
Wrong loop choice	Using `while` for a simple list	Use `for` for known collections
Premature break	Stopping before all required data are processed	Confirm that stopping is genuinely required
Over-skipping with continue	Skipping data that should be checked	Define invalid-data criteria clearly
Independent vs exclusive conditions	Multiple categories triggered unintentionally	Use if-elif-else for exclusive categories
Incorrect indentation	Condition/loop block misaligned	Indent consistently
Treating output as engineering truth	Ignoring assumptions and standards	Interpret results within the stated model and specification

22. A Complete Problem-Solving Template for Students

```
# 1. Input
data = [...]

# 2. Engineering parameters
limit = ...

# 3. Initial result
count = 0
total = 0

# 4. Repetition
for value in data:

    # 5. Validation / exclusion
    if value_is_invalid:
        continue

    # 6. Engineering decision
    if condition:
        count = count + 1
        total = total + value

# 7. Output
print("Count:", count)
print("Total:", total)
```

Students should adapt this template only after identifying the actual engineering logic. Do not copy the structure blindly.

23. Debugging and Verification Strategy

29. Test with one simple value where the expected answer is obvious.
30. Test one value exactly at the limit.
31. Test one value just below the limit.
32. Test one value just above the limit.
33. Test an empty or very small data set when appropriate.
34. Print intermediate values during development.
35. Check the number of loop iterations.

36. Check units and conversion factors.
37. Compare Python results with a hand calculation or spreadsheet for a small test case.
38. Only after verification should the program be applied to larger data.

24. Engineering Interpretation of Program Results

A Python program produces a computational result, not automatically a final engineering decision. The engineer must consider input quality, units, assumptions, model validity, uncertainty, code provisions and project-specific requirements.

- A Boolean `True` means only that the coded condition evaluated to True.
- A count of samples satisfying a criterion does not by itself establish code compliance unless the required acceptance procedure is satisfied.
- A rainfall threshold exceedance is not automatically a flood prediction.
- A survey value outside a tolerance should trigger the appropriate quality-control or adjustment procedure.
- A structural load calculation in a teaching program is not a substitute for a complete code-based structural analysis.

25. Mini Case Study: Processing Daily Rainfall

Problem statement: For seven daily rainfall observations, count warning days, calculate total valid rainfall, and stop the scan if a critical rainfall value above 200 mm is encountered.

```
rainfall = [45, 110, -1, 80, 155, 220, 90]
warning_limit = 100
critical_limit = 200

warning_days = 0
total = 0

for value in rainfall:
    if value < 0:
        continue

    if value > critical_limit:
        print("Critical rainfall detected:", value, "mm")
        break

    total = total + value

    if value > warning_limit:
        warning_days = warning_days + 1

print("Valid rainfall accumulated:", total, "mm")
print("Warning observations:", warning_days)
```

Sample output:

```
Critical rainfall detected: 220 mm
Valid rainfall accumulated: 390 mm
Warning observations: 2
```

Interpretation: The program skips the invalid -1 value, accumulates valid values until the first critical event, counts warning values encountered before that event, and then terminates. If the engineering requirement is to analyze all days, `break` would not be appropriate.

26. Mini Case Study: Construction Activity Screening

Problem statement: For several construction activities, report quantities above 20 m³ and calculate the total of all quantities.

```
activities = ["Footing", "Column", "Beam", "Slab", "Retaining Wall"]
quantities = [18, 12, 25, 32, 22]

total = 0

for i in range(len(activities)):
    total = total + quantities[i]

    if quantities[i] > 20:
        print(activities[i], "requires detailed quantity review")

print("Total quantity:", total, "m3")
```

Sample output:

```
Beam requires detailed quantity review
Slab requires detailed quantity review
Retaining Wall requires detailed quantity review
Total quantity: 109 m3
```

Interpretation: The loop performs accumulation and the condition flags selected activities. The review threshold is a planning example and should be defined by the project workflow.

27. Summary Decision Table

If the problem says...	Think of...
Check whether	`if`
Otherwise	`else`
Several possible categories	`elif`
For every observation	`for`
Until a condition is met	`while`
For every station and every observation	Nested loops
Stop immediately when	`break`
Ignore this record and continue	`continue`
Calculate a total	Accumulator inside a loop
Count qualifying observations	Counter inside a loop

28. Practice Section

A. Short-Answer Questions

39. What is the role of a condition in an engineering program?

40. What is the role of a loop?

41. When is a `for` loop generally suitable?
42. When is a `while` loop generally suitable?
43. What is meant by a threshold condition?
44. What is an accumulator?
45. What is a counter?
46. Why is indentation important in Python?
47. What is a nested loop?
48. Why should units be checked before applying a condition?

B. Conceptual Questions

49. Explain why loops and conditions are often used together in engineering data processing.
50. How would you decide whether a concrete-strength problem needs `if` or `if-else`?
51. Why is `if-elif-else` useful for classification problems?
52. Compare using a `for` loop with using a `while` loop for iterative engineering calculations.
53. Explain how `continue` can be used for invalid observations.
54. Explain when `break` is appropriate in a monitoring problem.
55. Why can a Python result be computationally correct but still not be an adequate engineering decision?
56. Explain the role of pseudocode before writing a Python program.
57. Why are boundary values such as exactly the permissible limit important when testing an engineering program?
58. Explain how nested loops can represent station $\times$ observation data.

C. Python Programming Exercises

59. Write a program to count values greater than a given threshold.
60. Write a program to classify a list of values into three ranges using if-elif-else.
61. Write a program to calculate the total of a list using a for loop.
62. Write a program to calculate the average of valid observations using a loop and condition.
63. Write a program to skip negative values using continue.
64. Write a program to stop processing when a value exceeds a critical threshold using break.
65. Write a while loop that repeats until an error becomes smaller than a specified tolerance.
66. Write a nested loop to process three stations with four observations at each station.
67. Write a program that reports both the count and total of observations satisfying a criterion.
68. Write a program combining validation, classification and accumulation for a list of engineering measurements.

D. Civil Engineering-Based Programming Problems

69. Concrete testing: Read several compressive-strength results, classify each as satisfying/not satisfying a minimum criterion, and report the count satisfying it.
70. Rainfall analysis: Process daily rainfall observations, skip invalid values, count warning observations and report the first critical observation.
71. Surveying: Check measured distances against a target and tolerance and report each observation as within/outside tolerance.
72. Construction quantity estimation: Process quantities for multiple activities, calculate total quantity and identify activities above a selected review threshold.
73. Structural load processing: Use nested loops to calculate total load for several load cases and flag cases above an illustrative screening limit.

E. Viva Questions with Answers

Question	Answer
Why are loops useful in Civil Engineering programs?	They automate repetitive calculations for observations, samples, stations, activities or cases.
Why are conditions useful?	They allow a program to make decisions based on engineering criteria.
Which loop is convenient for a known list of observations?	Usually a `for` loop.
Which loop is useful when repetition continues until a condition changes?	A `while` loop.
What is the purpose of combining a loop with `if`?	To process multiple values and make a decision for each value.

29. Faculty Teaching Notes

74. Start with a paper-based engineering problem before showing code.
75. Ask students to underline words such as 'each', 'every', 'until', 'if', 'minimum', 'maximum', 'within', and 'exceeds'.
76. Map 'each/every' to repetition and 'if/unless/otherwise' to conditions.
77. Write the algorithm on the board before the Python code.
78. Trace one data item through the loop and condition.
79. Use boundary-value examples: exactly at, just below and just above the limit.
80. Ask students to interpret the result in engineering language rather than merely reading the Python output.
81. Emphasize that illustrative thresholds are not design-code requirements.

30. Lecture Takeaway

The central skill in engineering programming is not memorizing Python syntax; it is translating an engineering procedure into a clear sequence of decisions and repetitions. Use conditions when the program must decide, loops when it must repeat, and combine them when many engineering observations must each be evaluated. With this approach, simple Python programs can automate repetitive Civil Engineering tasks such as data screening, threshold checking, classification, quantity accumulation, survey tolerance checks and iterative calculations. The engineer remains responsible for selecting valid data, assumptions, units, methods and applicable standards.