FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XI · CODE 308

Flow of Control

CUET unit: Flow of Control

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- **Flow of control** is the order in which statements in a Python program are executed; it can be sequential, selective, or repetitive.
- **Selection** (`if` , `if..else` , `if..elif..else`) is the mechanism for decision-making in Python, replacing the default top-to-bottom sequence when a condition must be evaluated.
- **Indentation** is a first-class syntactic rule in Python (unlike curly-bracket languages), making it directly testable in CUET.
- **Repetition** through `for` and `while` loops — including the `range()` function, `break` , `continue` , and nested loops — is the most MCQ-heavy portion of this topic.
- CUET tests this area heavily for output-tracing, syntax identification, and choosing the correct loop/control construct for a described scenario.

📖 Detailed Notes

2.1 Core concepts

- **Flow of control** is defined as the order of execution of statements in a program. It can be implemented using control structures. Python supports two types of control structures: **selection** and **repetition**. The default execution order (statement by statement from top to bottom) is called **sequence**. (NCERT §6.1, p. 121)
- **Selection** (`if..else`) is used when a decision must be made between two or more alternative paths. The concept of selection is implemented using the `if..else` statement in Python. (NCERT §6.2, p. 122)
- The syntax of the simple `if` statement is: `if condition:` followed by an indented block of `statement(s)` . If the condition is true, the indented statements execute; if false, they are skipped. (NCERT §6.2, p. 123)
- The `if..else` statement allows two alternative blocks: the `if` block executes when the condition is true, and the `else` block executes when the condition is false. (NCERT §6.2, p. 123)
- The `if..elif..else` chain (where `elif` means `else..if`) is used when multiple conditions must be checked in sequence. As soon as one condition is true, its indented block executes and the rest of the `if` statement terminates. The number of `elif` clauses depends on the number of conditions. (NCERT §6.2, p. 124)

- A nested `if` is an `if..else` placed inside another `if` or `elif` block. Python allows many levels of such nesting. (NCERT §6.2, p. 126)
- **Indentation** in Python refers to the leading whitespace (spaces or tabs) at the beginning of a statement. The same level of indentation associates statements into a single block of code. Python's interpreter checks indentation strictly and raises a syntax error if it is incorrect. The common practice is to use a single tab for each level. Unlike most other languages, Python does **not** use curly brackets for blocks. (NCERT §6.3, p. 126)
- **Repetition** (also called **iteration**) means executing a set of statements repeatedly. This is implemented using looping constructs. Python provides two looping constructs: `for` and `while`. The condition is checked based on the value of a **control variable**. When the condition becomes false, the loop terminates. The programmer must ensure an exit condition exists; otherwise an **infinite loop** results. (NCERT §6.4, p. 127–128)
- The `for` **loop** iterates over a range of values or a sequence (numeric values, characters of a string, elements of a list or tuple). It is used when the number of iterations is known in advance. Syntax: `for <control-variable> in <sequence/range>:` followed by the indented body. When all items in the range are exhausted, control transfers to the statement after the loop. (NCERT §6.4.1, p. 128)
- The `range()` **function** is a built-in Python function with syntax `range([start], stop[, step])`. It generates a sequence of integers from `start` up to (but not including) `stop`, incrementing by `step`. Default `start` is 0; default `step` is 1. `step` can be a positive or negative integer but not zero. All parameters must be integers. (NCERT §6.4.1(B), p. 130)
- The `while` **loop** executes its body repeatedly as long as the control condition is true. The condition is evaluated before each iteration. If the condition is initially false, the body is not executed even once. The body must contain statements that eventually make the condition false to avoid an infinite loop. (NCERT §6.4.2, p. 131)
- The `break` **statement** immediately exits (terminates) the current loop and resumes execution at the statement following the loop body. It alters the normal flow of execution inside a loop. (NCERT §6.5.1, p. 133)
- The `continue` **statement** skips the remaining statements of the current iteration and jumps back to the beginning of the loop for the next iteration. Unlike `break`, `continue` does not exit the loop — it only skips the rest of the current pass. (NCERT §6.5.2, p. 135)
- **Nested loops**: A loop contained within another loop is called a nested loop. Python imposes no restriction on how many loops can be nested or on levels of nesting. Any combination of `for` and `while` loops can be nested. For each single iteration of the outer loop, the inner loop runs through its complete cycle. (NCERT §6.6, p. 136–137)

2.2 Definitions to memorise

Term	Definition	Page
Flow of control	The order of execution of statements in a program	121
Sequence	Default execution order — one statement after another from beginning to end	121
Selection	A control structure that chooses between two or more alternative paths based on a condition; implemented using `if..else`	122
Indentation	Leading whitespace (spaces/tabs) at the beginning of a statement; used by Python to define blocks of code	126
Repetition / Iteration	Executing a set of statements repeatedly using a loop construct	127
Control variable	The variable whose value is checked to determine whether a loop continues or terminates	128
Infinite loop	A loop that never terminates because the condition never becomes false; a logical error	128
`for` loop	A looping construct that iterates over each item in a range or sequence; number of iterations is known in advance	128
`range()`	A built-in Python function that generates a sequence of integers; syntax: `range([start], stop[, step])`	130
`while` loop	A looping construct that executes its body as long as the test condition remains true; number of iterations need not be known in advance	131
`break` statement	Immediately exits the current loop; execution resumes after the loop body	133
`continue` statement	Skips the remaining statements of the current iteration and jumps to the next iteration of the loop	135
Nested loop	A loop contained inside another loop	136
`if` statement	Simple selection construct that executes a block when its condition is True	123
`if..else` statement	Two-way selection executing one block on True and another on False	123
`if..elif..else` statement	Multi-way selection that checks several conditions in sequence	124
Nested `if`	An `if..else` placed inside another `if` or `elif` block	126
Block	A group of statements sharing the same indentation level	126
`elif`	Python keyword meaning "else if"; used in multi-way selection	124
`pass` statement		122

Term	Definition	Page
	Null statement used as a placeholder where syntactically a statement is required	
Step parameter (`range`)	Increment between successive values in a `range()` ; may be positive or negative but not zero	130
Stop value (`range`)	Exclusive upper limit of `range()` ; the generated sequence stops before this value	130
Start value (`range`)	Inclusive lower limit of `range()` ; default is 0 if omitted	130
Counter-controlled loop	Loop driven by a counter variable typical of `for` constructs	128
Condition-controlled loop	Loop whose iteration depends on a Boolean test typical of `while`	131

2.3 Diagrams / processes to remember

- **Figure 6.2 (p. 122):** Flowchart depicting decision making — shows a diamond (decision) shape with `num1 > num2?` ; the "Yes" branch computes `diff = num1 - num2` , the "No" branch computes `diff = num2 - num1` . Illustrates the `if..else` structure visually.
- **Figure 6.4 (p. 128):** Flowchart of the `for` loop — Start → Initialisation → Test Expression → (True) Body of For Loop → back to Test Expression; (False) → Exit for Loop → Statement following the loop → Stop.
- **Figure 6.5 (p. 131):** Flowchart of the `while` loop — Start → Initialisation → Test Expression → (True) Body of while Loop → back to Test Expression; (False) → Statements following while loop → Stop.
- **Figure 6.5 / break flowchart (p. 133):** Flowchart showing `break` — within the loop body, if the break condition is encountered, control immediately exits the loop to the statement following it.
- **Figure 6.6 (p. 135):** Flowchart of `continue` — when `continue` is encountered, control jumps back to the loop condition check (not to the statement after the loop), skipping remaining statements of that iteration.

2.4 Common confusions / NTA trap points

- `break` **vs** `continue` **:** Students confuse the two. `break` exits the loop entirely; `continue` only skips the rest of the current iteration and re-checks the loop condition. NTA frequently presents code snippets and asks what is printed — identifying whether `break` or `continue` is present is critical.
- `range()` **stop value is exclusive:** `range(1, 6)` produces 1, 2, 3, 4, 5 — NOT 6. NTA constructs distractors that include the stop value in the output. Also, `range(10)` starts at 0, not 1.

- `while` **loop with initially-false condition:** If the condition is false from the start, the body executes zero times. NTA asks "how many times is the body executed?" for such cases.
- `elif` **vs** `else` **:** `elif` checks a new condition; `else` catches all remaining cases. Using `elif condition` without a final `else` means nothing executes if all conditions are false — NTA tests this with output-tracing questions.
- **Indentation errors (NCERT §6.3, p. 126).** A statement with wrong indentation belongs to a different block. NTA may show code where a `print` is at the wrong indent level inside a loop or `if`, and ask for the output — the answer changes entirely based on indentation.
- `range()` **step cannot be zero (NCERT §6.4.1(B), p. 130).** `range(0, 5, 0)` raises ValueError. NTA distractor may claim step=0 produces an infinite sequence.
- `for` **loops do NOT need to know iteration count at compile time (NCERT §6.4.1, p. 128).** They iterate over any sequence; the programmer must know the iteration count, but Python figures it out at runtime.
- `break` **only exits the innermost loop (NCERT §6.5.1, p. 133).** In nested loops, `break` does not exit all enclosing loops. NTA traps assume `break` "exits all loops".
- `else` **clause on loops is outside the NCERT syllabus here.** Stick to NCERT — `for..else` / `while..else` are not covered. Avoid that in CUET answers.
- **Negative step requires start > stop (NCERT §6.4.1(B), p. 130).** `range(0, 9, -1)` yields an empty sequence; `range(9, 0, -1)` yields 9, 8, ..., 1.
- **Iteration variable persists after the loop (NCERT §6.4.1).** After `for i in range(5):`, `i` is 4 outside the loop body. NTA may use this in output-tracing.

🎯 Practice MCQs

Q1. What will be the output of the following Python code? ```python for i in range(2, 10, 3): print(i, end=' ') ```

A. 2 5 8 11

B. 2 4 6 8

C. 2 5 8

D. 3 6 9

Q2. Which of the following statements about indentation in Python is correct?

A. Python uses curly brackets `{}` to define a block of code, and indentation is optional.

B. Leading whitespace at the beginning of a statement is called indentation; Python uses it to define blocks, and incorrect indentation raises a syntax error.

C. Indentation is only required inside `for` loops, not inside `if` statements.

D. Python allows any number of spaces for indentation within the same block, as long as each statement is indented by at least one space.

Q3. Consider the following code segment: ```python num = 0 for num in range(10): num = num + 1 if num == 8: break print('Num has value ' + str(num)) print('Encountered break!! Out of loop') ``` How many times is `'Num has value ...'` printed?

A. 10

B. 9

C. 8

D. 7

📊 PYQ Alignment

Flow of control is consistently tested in CUET Computer Science papers, typically contributing 4–6 questions per year; questions most often require tracing the output of `for` / `while` loop code snippets (especially those involving `range()`, `break`, or `continue`), identifying correct syntax for `if..elif..else` chains, and distinguishing the behaviour of `break` versus `continue` in nested loops. See the PYQ archive for Computer Science for more practice.