

TOPIC 1.4: CONDITIONAL STATEMENTS IN TEXT-BASED PROGRAMMING

Comprehensive Classroom Notes & Pedagogical Scaffold | Grade 7 ICT (Python)

Subject: Information Communication Technology	Unit: 1. Computational Thinking & Programming
Level: Grade 7 / Level 7	Topic Focus: 1.4 Python IF, ELIF, ELSE & Boolean Logic

1. Learning Objectives

By the end of this topic, students will be able to:

- Explain how conditional (selection / branching) statements direct the execution path of a text-based Python program.
- Understand and write Python comparison operators (>, <, >=, <=, ==, !=).
- Construct single-branch (if), dual-branch (if-else), and multi-branch (if-elif-else) selection statements with strict syntax adherence.
- Apply whitespace indentation (using the Tab key) correctly to define statement blocks and explain why Python does not use ENDIF.
- Combine multiple boolean conditions using Python logical operators (and, or, not).
- Systematically debug Python selection code to fix syntax, indentation, type casting, and logical ordering errors.

2. Classroom Discussion of Core Theoretical Concepts

A. Nature of Conditional Statements:

Conditional statements (also called selection statements or branching statements) allow programs to make decisions. They test a condition that evaluates strictly to True or False. If True, one branch of code executes; if False, another branch (or nothing) executes.

B. Python Comparison / Relational Operators:

Python uses six standard relational operators for evaluating numerical and string expressions:

- > : Greater than (e.g., 10 > 2 -> True)
- < : Less than (e.g., 1 < 2 -> True)
- >= : Greater than or equal to (e.g., 100 >= 100 -> True)
- <= : Less than or equal to (e.g., 82 <= 99 -> True)
- == : Equal to (e.g., colour == 'purple' -> True) [CRITICAL: Do not confuse with assignment =. The double equals checks equality!]
- != : Not equal to (e.g., 2 != 1 -> True)

C. Structure of Python Selection Statements:

1. Single Branch (if):

```
if condition:
    [indented code that runs when True]
```

2. Dual Branch (if-else):

```
if condition:
    [indented code when True]
else:
```

```
`[indented code when False]`
```

3. Multi-Branch (`if-elif-else`):

```
`if condition1:`  
    `[code if condition1 is True]`  
`elif condition2:`  
    `[code if condition1 is False AND condition2 is True]`  
`else:`  
    `[code if ALL preceding conditions are False]`
```

D. The Role of Indentation in Python:

Unlike pseudocode which uses `ENDIF`, Python relies entirely on indentation (whitespace to the right) to know which lines belong inside an `if`, `elif`, or `else` block. Indentation is created by pressing the Tab key or 4 spaces. Unindented code sits outside the conditional and will always run regardless of whether the condition was True or False.

E. Python Boolean Operators (`and`, `or`, `not`):

1. `and`: Both conditions on either side must be True for the entire statement to be True (e.g., `age >= 18 and type == 'Drama'`).
2. `or`: At least one condition (or both) must be True. It is False only if both conditions are False (e.g., `ans == 'yes' or ans == 'YES'`).
3. `not`: Reverses / inverts the truth value of a condition (e.g., `not (guess == 'red')` is True if guess is anything other than 'red').

F. Classic Python Syntax & Runtime Errors (Table 1.6):

- Uppercase Keywords: Writing `IF`, `ELSE`, `ELIF`, `AND`, `OR`, `NOT` causes syntax errors; Python keywords are strictly lowercase.
- Single Equals Bug: Writing `if score = 50:` (syntax error) instead of `if score == 50:`.
- Missing Colon: Omitting `:` at the end of an `if`, `elif`, or `else` line.
- Indentation Mismatch: Forgetting to indent inside a block or failing to backspace when writing `elif`/`else`.
- Misplaced `else`: Placing `elif` after an `else` block (`else` must always be the final fallback).
- Multiple `else` statements: Having more than one `else` in a single conditional construct.

3. Question and Answers

Q1: What is the primary command word for a conditional statement in Python?

Answer: The lowercase keyword `if`.

Q2: How does Python determine which lines of code belong inside an IF statement?

Answer: By looking at the indentation (whitespace to the right created with the Tab key). Python does not use `ENDIF`.

Q3: What is the difference between `=` and `==` in Python?

Answer: `=` is the assignment operator used to store a value in a variable (e.g., `total = 10`), whereas `==` is a comparison operator used to test whether two values are equal (e.g., `if total == 10`).

Q4: How many ELSE statements can a single conditional block contain in Python?

Answer: Exactly one (or none). A conditional construct cannot have multiple `else` statements, and the `else` must always be the final block.

Q5: In an IF-ELIF-ELSE chain, when does the second condition (the ELIF) get evaluated?

Answer: The second condition is tested only when the preceding `if` condition evaluates to False.

Q6: Look at this script: ``val = 10; if val < 10: print('Tortoise') else: print('Hamster')``. What is printed?

Answer: ``Hamster``. Because 10 is not strictly less than 10, the condition is False, causing the ``else`` branch to execute.

Q7: What does the Boolean operator ``and`` require for a compound condition to evaluate to True?

Answer: It requires both conditions (on the left and right side of ``and``) to evaluate to True.

Q8: Why does ``input('Enter age: ')`` require ``int()`` when used with ``>= 18`` in Python?

Answer: Because ``input()`` always returns a string (text). Attempting to compare a string with an integer using ``>=`` causes a ``TypeError``. ``int()`` converts the text to a number.

4. Second-Order Thinking & Analytical Scenarios

SO-Q1 [Sequential Branching & Logic Flaws]:

Scenario Analysis: Consider a student grading algorithm written in Python:

```
`score = int(input('Enter score: '))`  
`if score >= 50: print('Bronze')`  
`elif score >= 60: print('Silver')`  
`elif score >= 80: print('Gold')`  
`else: print('Fail')`
```

Evaluate what output a student with a score of 95 receives, and explain the architectural flaw in the condition order.

- Output: The program outputs ``Bronze``!
- Analysis: Python executes ``if-elif`` chains sequentially. Since ``95 >= 50`` is True, Python executes the first branch and immediately exits the structure, skipping all subsequent checks. To fix this logic error, conditions must be ordered from highest threshold to lowest (``>= 80`` first, then ``>= 60``, then ``>= 50``).

SO-Q2 [Boolean Operator Precedence & Grouping]:

Scenario Analysis: A smart-home security alarm has two sensors: ``door_open`` (True/False) and ``motion_detected`` (True/False). The alarm should trigger if either the door is opened OR motion is detected, but ONLY IF the ``system_armed`` is kept as TRUE by the user.

- Evaluate the difference between these two Python statements:

Statement 1: ``if system_armed and door_open or motion_detected:``

Statement 2: ``if system_armed and (door_open or motion_detected):``

- Evaluation: In Statement 1, due to operator precedence, motion detected while the system_alarm is FALSE will still trigger the alarm because ``or motion_detected`` sits outside the ``and``.

Statement 2 correctly uses parentheses to enforce that the system_alarm must be TRUE before evaluating sensor inputs.

5. Key Vocabulary Glossary

Key Term	Textbook Definition & Classroom Context
Conditional Statement	A programming statement (such as IF) that runs specific instructions based on whether a condition is True or False.

Selection Statement	Another name for a conditional statement, because it selects which branch of code to run.
Branching	The process of dividing program execution into two or more distinct paths based on a decision.
Indentation	The number of spaces (or Tab space) by which a line of code is moved to the right to show it belongs inside a statement.
Relational Operator	Symbols used to compare values in Python (>, <, >=, <=, ==, !=).
elif	Python's shorthand keyword for ELSEIF, used to test secondary conditions in a multi-branch chain.
Boolean and	A logical operator requiring both conditions to be True for the overall expression to be True.
Boolean or	A logical operator requiring at least one condition to be True for the overall expression to be True.