If Else Class thoery

Python Conditions and If Statements

1. Introduction

Conditional statements enable a program to execute certain lines of code only when specific conditions are met. In Python, these conditions are typically comparisons or logical checks that result in `True` or `False` .

Common Comparison Operators

- `==` (Equals)
  Checks if two values are the same.
  Example: `a == b`
- `!=` (Not Equals)
  Checks if two values are different.
  Example: `a != b`
- `<` (Less Than)
  Checks if `a` is strictly less than `b` .
  Example: `a < b`
- `<=` (Less Than or Equal To)
  Checks if `a` is less than or equal to `b` .
  Example: `a <= b`
- `>` (Greater Than)
  Checks if `a` is strictly greater than `b` .
  Example: `a > b`
- `>=` (Greater Than or Equal To)
  Checks if `a` is greater than or equal to `b` .
  Example: `a >= b`

These checks (conditions) are often used with **if statements** and loops.

2. Basic If Statements

A simple `if` statement checks one condition. If the condition is `True` , the indented code block runs; otherwise, it’s skipped.

Syntax

```
if condition:
    # code to run if condition is True
```

Example

```
a = 33
b = 200

if b > a:
    print("b is greater than a")
```

Here, `b > a` evaluates to `True`, so “b is greater than a” is printed.

Indentation

In Python, **indentation** defines the scope of the code block. Other languages may use curly braces `{}`, but Python uses whitespace. Incorrect indentation causes an error:

```
if b > a:
print("b is greater than a")  # This will raise an indentation error
```

3. `elif` (Else If)

Use `elif` for additional conditions after an `if`. Python evaluates each in sequence and stops when one is `True`.

Syntax

```
if condition1:
    # block1
elif condition2:
    # block2
else:
    # block3
```

Example

```
a = 33
b = 33

if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
else:
    print("a is greater than b")
```

- Since `a == b`, the second condition is `True`, and we print “a and b are equal.”

4. `else` Statement

`else` handles any case not caught by previous `if` or `elif`.

```
a = 200
b = 33

if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

- `a` is greater than `b`, so we end up in the `else` branch.

You can also use an `else` directly after an `if` without an `elif`:

```
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

5. Short-Hand If and Ternary Operators

Short-Hand If

If the code block is a single statement, it can follow the `if` on the same line:

```
if a > b: print("a is greater than b")
```

Short-Hand If ... Else (Ternary)

A one-line conditional can be used to handle a quick decision:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

You can chain multiple conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

6. Logical Operators: `and` , `or` , `not`

`and`

Both conditions must be `True` .

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

`or`

At least one condition must be `True` .

```
if a > b or a > c:
    print("At least one of the conditions is True")
```

`not`

Reverses the result of a conditional check.

```
if not a > b:
    print("a is NOT greater than b")
```

7. Nested If Statements

An `if` statement can appear inside another `if` block, creating multiple layers of checks:

```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

- This code checks `x > 10` first, then (if true) checks `x > 20`.

8. The `pass` Statement

`if` statements cannot be empty. Use `pass` as a placeholder if you need a syntactically valid block without actual code:

```
a = 33
b = 200

if b > a:
    pass  # do nothing for now
```

9. Exercise

1. **Check Equality**
   - Prompt the user for two numbers, say `x` and `y`. Print whether they are equal, or which is larger.
2. **Even/Odd**
   - Ask a user for a number. Use `if-else` to determine if it's even (`num % 2 == 0`) or odd.
3. **Nested Condition**
   - If `age` ≥ 18, print "Adult." Inside that block, check if `age` ≥ 65, print "Senior Citizen."

Python's conditional statements (`if` , `elif` , `else`) allow your program to branch based on true/false conditions. **Logical operators** (`and` , `or` , `not`) let you combine multiple criteria. Mastering these concepts is crucial for building programs that adapt to user input or real-world data.

Remember to keep a close eye on **indentation** in Python unlike other languages that use braces, Python's indentation is part of the language syntax. Proper understanding and usage of these features will significantly enhance your ability to create flexible and intelligent applications.