Python Notes – Conditionals (Core Concepts)

Data Types Recap

- Python has different data types (numbers, strings, dates, etc.).
- We process data using **logic**.

Conditionals

- Used to make **decisions** in code.
- Condition = must evaluate to **True** or **False** (Boolean).
- Keyword: `if`

Boolean Values

- `True` or `False`

Example:

```
kettle_boiled = True
```

-

If Statement

Syntax:

```
if condition:
    # code runs if condition is True
```

-

Example:

```
if kettle_boiled:
    print("Kettle done! Time to make chai.")
```

-

Important Points

- Use `:` after `if`.
- **Indentation** matters (Python uses spaces to mark blocks).
- If condition is False → code block is skipped.

Python Notes – Snack Suggestion Project (Core Concepts)

Problem

- Local café wants a **snack suggestion/ordering system**.
- User enters a snack → program checks if it’s available (cookies or samosa).

Input in Python

- Use `input()` to take user input (from command line).

Example:

```
snack = input("Enter your preferred snack: ")
```

-
- Input is always a **string** (even numbers).

String Normalization

- Users may type in different cases (`Burger`, `burger`, `BURGER`).

Use `.lower()` to make input consistent:

```
snack = snack.lower()
```

-

Comparison Operators

- `=` → assignment (store value in a variable).
- `==` → comparison (check if values are equal).

Conditional Logic

- If input is **cookies** or **samosa**, confirm order.
- Otherwise, show unavailable message.

Example:

```
if snack == "cookies" or snack == "samosa":
    print(f"Great choice! We will serve you {snack}.")
else:
    print("Sorry, we only serve cookies or samosa.")
```

Indentation

- Python uses **indentation** (spaces) to mark code blocks.
- Always put `:` after `if` or `else`.

✅ **Core takeaway:**

- Use `input()` for user input.
- Normalize strings with `.lower()`.

- Use `==` for comparison.
- Build decision-making using `if...else`.

Python Notes – Chai Price Calculator (Core Concepts)

Problem

- A tea stall sells cups in **3 sizes**: small, medium, large.
- Program should calculate the price based on size.
- If input is invalid → show "Unknown cup size".

Input

- Use `input()` to take user choice.

Convert to lowercase for consistency:

```
cup = input("Choose cup size (small/medium/large): ").lower()
```

-

Conditional Statements

- Use **if / elif / else** for multiple conditions.

Example:

```
if cup == "small":
    print("Price is 10 rupees")
elif cup == "medium":
    print("Price is 15 rupees")
```

```
elif cup == "large":
    print("Price is 20 rupees")
else:
    print("Unknown cup size")
```

Key Concepts

- `if` → first condition.
- `elif` → check multiple conditions.
- `else` → fallback (for all other cases).
- Normalize input with `.lower()`.
- Indentation and `:` are mandatory.

✅ **Core takeaway:**

- Use `if / elif / else` for **multi-condition decisions**.
- Always handle invalid inputs with `else`.

Python Notes – Smart Thermostat Alert System (Core Concepts)

Problem

- Build a system that checks:
    1. **Device status**
        - If **active** → check temperature.

        - If **offline** → print “Device is offline”.

2. **Temperature**
    - If **> 35°C** → print “High temperature alert!”.
    - Else → print “Temperature is normal”.

Nested If (Nesting)

- An **if inside another if** is called **nested if**.
- Useful when one condition depends on another.

Example Code

```
device_status = "active"
temperature = 38

if device_status == "active":
    if temperature > 35:
        print("High temperature alert!")
    else:
        print("Temperature is normal")
else:
    print("Device is offline")
```

Key Concepts

- **Nesting** = if statements inside another if.
- Each `else` belongs to the nearest matching `if`.
- **Indentation** defines which block an else/if belongs to.

- `pass` keyword can be used as a **placeholder** (do nothing temporarily).

✅ **Core takeaway:**

- Use nested if when conditions depend on each other.
- Indentation is critical for readability and correctness.

Python Notes – Smart Thermostat Alert System (Core Concepts)

Problem

- Build a system that checks:
  - **Device status**
    - If `active` → check temperature.
    - If `offline` → print `"Device is offline"`.
  - **Temperature**
    - If `> 35°C` → print `"High temperature alert!"`.
    - Else → print `"Temperature is normal"`.

Nested If (Nesting)

- An **if inside another if** = **nested if**.
- Used when **one condition depends on another**.

Example Code

```
device_status = "active"
temperature = 38

if device_status == "active":
    if temperature > 35:
        print("High temperature alert!")
    else:
        print("Temperature is normal")
else:
    print("Device is offline")
```

Key Concepts

- **Nesting** → if statements inside another if.
- Each `else` → belongs to the **nearest if**.
- **Indentation** → defines which block code belongs to.
- `pass` keyword → placeholder (do nothing temporarily).

✅ Core takeaway

- Use **nested if** when conditions depend on each other.
- Proper **indentation** is critical for readability and correctness.

Python Notes – Train Seat Classification (Core Concepts)

Problem

- Build a **ticket info system** for a railway app.
- Based on **seat type**, show features.
- Seat categories:
    - `sleeper`
    - `ac`
    - `general`
    - `luxury`
- If seat type doesn't match → show `"Invalid seat type"`.

Match–Case (Alternative to If–Else)

- When there are many cases, **match–case** is easier and more readable than long if–elif chains.

Syntax:

```
match variable:
    case value1:
        # action
    case value2:
        # action
    case _:
        # default action
```

-
- `_` → wildcard (matches anything else).

Example Code

```
seat_type = input("Enter seat type (sleeper, ac, general, luxury): 
").lower()

match seat_type:
    case "sleeper":
        print("No AC, Beds available")
    case "ac":
        print("Air conditioned, Comfy ride")
    case "general":
        print("Cheapest option, No reservation")
    case "luxury":
        print("Premium seats with meals")
    case _:
        print("Invalid seat type")
```

Key Concepts

- **`match`** checks variable against multiple possible values.
- **`case`** handles each specific match.
- **`case _`** acts like **else** (default).
- `.lower()` makes input case-insensitive.

✅ Core takeaway

- Use **match–case** when handling multiple categories.
- Improves readability compared to long if–elif–else chains.