1. Different types of numbers in Python

In programming, numbers aren't all the same — they're categorized into types.

- **Integers (`int`)**
  Whole numbers (no decimal).
  Example: `5`, `-42`, `2025`

- **Booleans (`bool`)**
  Special type of numbers that can only be `True` or `False`.
  Behind the scenes, Python treats `True` as `1` and `False` as `0`.
  Example: `is_logged_in = True`

- **Floating point numbers (`float`)**
  Numbers with decimals (used when precision is important).
  Example: `3.14`, `2.0`, `-0.75`
  Useful for things like money, stock prices, temperature.

- **Complex numbers (`complex`)**
  Numbers with a real and an imaginary part (rarely used outside math/science).
  Example: `2 + 3j`

---

2. Variables & Operations

A **variable** is just a name that stores a value in memory.
For example:

```
black_tea_grams = 14
ginger_grams = 3
total_grams = black_tea_grams + ginger_grams
print(total_grams)   # Output: 17
```

➡️ Here, you're using **addition (`+`)**. Similarly, you can use:

- Subtraction: `-`

- Multiplication: `*`
- Division: `/`

---

3. Division in Python

This is where many beginners get confused.
Python gives us **two types of division**:

True Division (`/`)
Always gives a decimal (float).

```
7 / 4   # 1.75
```

-

Floor Division (`//`)
Ignores everything after the decimal, keeps only the whole number.

```
7 // 4  # 1
```

-

Think of `//` as "I only care about full units, ignore leftovers."

---

4. Modulo Operator (`%`)

Used to find the **leftover** after division.
Example:

```
10 % 3   # 1 (because 10 = 3*3 + 1)
```

In the transcript, when they wanted to find leftover cardamom pods, they're basically using `%`.

---

5. Formatted Strings (f-strings)

This is how you insert variable values into a string:

```
total = 17
print(f"Total grams of tea is {total}")
```

Output:

```
Total grams of tea is 17
```

✅ So summarizing the **big concepts here**:

1. Python number types: `int`, `bool`, `float`, `complex`
2. Variables store values in memory.
3. Operations: `+`, `-`, `*`, `/`, `//`, `%`
4. `//` means floor division (whole number only).
5. `%` gives leftovers (remainder).
6. `f""` strings let you mix variables into text.

1. Boolean basics

- Boolean is a data type with only **two values**:
    - `True` ✅
    - `False` ❌

Example:

```
is_boiling = True
```

```
print(is_boiling)   # True
```

Booleans answer **yes/no questions** like:

- Is the user logged in?
- Is the temperature above 42°C?
- Is there milk in the shop?

---

2. Boolean = Numbers

- In Python, `True` is stored as **1**
- `False` is stored as **0**

This means you can use them in math:

```
stir_count = 5
is_boiling = True

total_actions = stir_count + is_boiling
print(total_actions)   # 6 (because True = 1)
```

This conversion from Boolean → number is called **upcasting**.

---

3. Converting values to Boolean

You can use the `bool()` function to check the truthiness of values:

```
print(bool(0))       # False
print(bool(1))       # True
print(bool(11))      # True
print(bool("Hitesh")) # True (non-empty string)
print(bool(""))      # False (empty string)
```

```
print(bool(None))    # False
```

✅ Rule:

- **0, None, empty strings (""), empty lists ([ ]) → False**
- **Everything else → True**

---

4. Logical Operations

Python has 3 logical operators:

1. **and** → both must be true
2. **or** → at least one must be true
3. **not** → reverses true/false

Examples:

```
water_hot = True
tea_added = False

can_serve = water_hot and tea_added
print(can_serve)   # False (because tea not added)

can_buy = True or False
print(can_buy)     # True (at least one is true)

print(not True)    # False
print(not False)   # True
```

---

5. Example with chai shop

```
water_hot = True
tea_added = False
```

```
can_serve_chai = water_hot and tea_added
print(can_serve_chai)   # False → water is hot but no tea added

# After adding tea
tea_added = True
can_serve_chai = water_hot and tea_added
print(can_serve_chai)   # True → chai ready ✅
```

👉 So the main ideas are:

- Booleans are `True`/`False`
- They behave like `1` and `0`
- `bool()` can convert other values to Boolean
- Use `and`, `or`, `not` for logic

1. What are floating point numbers?

- They are numbers with a **decimal point**.
- Examples: `3.14`, `2.0`, `-7.25`, `95.5`
- In Python, they have the type `float`.

Why do we need them?

- Integers (`int`) can only represent whole numbers.
- But in real life, we often need fractions or decimals (like money, weight, scientific values).

---

2. Precision problem

Here's where it gets tricky:

- Computers store numbers in **binary (0s and 1s)**.
- Some decimal numbers cannot be represented **exactly** in binary.

Example:

```
print(0.1 + 0.2)   # 0.30000000000000004
```

➡️ You expected `0.3`, but the computer gives a slightly different answer.

This is because **floating points are an approximation**.

---

3. Example from transcript

Let's say you have:

```
x = 95.5
y = 95.49999999999999
print(x == y)   # False
```

Even though **to your eyes** they look almost the same, Python sees them as **different values** because of that small precision difference.

---

4. Why does this matter?

- In financial apps, scientific calculations, or anything that needs high accuracy, you can't always trust floats for equality checks.
- Instead of checking `x == y`, you check if they are **close enough**.

Python has a helper for this:

```
import math

x = 95.5
y = 95.49999999999999

print(math.isclose(x, y))   # True
```

➡️ `math.isclose()` checks if two numbers are nearly equal.

---

5. Special float values

Python also supports:

- `float("inf")` → infinity
- `float("-inf")` → negative infinity
- `float("nan")` → Not a Number (e.g., `0/0`)

Example:

```
print(1.0 / 0.0)   # Error: division by zero
print(float("inf")) # inf
```

✅ **Big concepts from floating points:**

1. Floats represent decimals (`3.14`, `95.5`).
2. They're stored in binary, so **not always exact**.
3. Equality (`==`) can fail due to precision issues.
4. Use `math.isclose(a, b)` to safely compare floats.
5. Special values: `inf`, `-inf`, `nan`.