✅ `>>> 9999999999 + 1`

Output: `10000000000`

- This shows Python's ability to handle very large integers without overflow.
- Python integers (`int`) can grow arbitrarily large.

⚠️ `>>> 9999999999 * 2.1`

Output: `20999999997.9`

- This result is **correct**, but here's the subtlety:
  - `9999999999` is an `int` , and `2.1` is a `float` .
  - The result becomes a `float` , so precision may be slightly limited for very large values.
  - Python uses IEEE 754 double-precision floats, which have about 15–17 digits of precision.
  - **So while this looks fine, at even larger scales, floating-point inaccuracies can occur.**

✅ `>>> 2 ** 200`

Output: A huge number

- Python handles large powers with big integers just fine.
- No overflow due to arbitrary-precision arithmetic with `int` .

✅ `>>> 2 + 1j`

Output: `(2+1j)`

- Creates a complex number (`complex` type).

✅ `>>> (2 + 1j) * 3`

Output: `(6+3j)`

- Multiplying complex numbers works naturally.

✅ `>>> 0o20`

Output: `16`

- `0o` prefix indicates **octal** (base-8).
- `0o20` in base 8 = `2 × 8 + 0 = 16`.

✅ `>>> 0xFF`

Output: `255`

- `0x` prefix indicates **hexadecimal** (base-16).
- `FF = 15×16 + 15 = 255`.

✅ `>>> 0b1000`

Output: `8`

- `0b` prefix indicates **binary**.
- `1000 = 8` in decimal.

✅ `>>> oct(64)`

Output: `'0o100'`

- Converts decimal 64 to **octal string** representation.

✅ `>>> hex(64)`

Output: `'0x40'`

- Converts 64 to **hexadecimal string**.

✅ `>>> bin(64)`

Output: `'0b1000000'`

- Converts 64 to **binary string**.

✅ `>>> int('64', 8)`

Output: `52`

- Interprets `'64'` as **base-8** → `6×8 + 4 = 52`.

✅ `>>> int('64', 16)`

Output: `100`

- Interprets `'64'` as **base-16** → `6×16 + 4 = 100`.

✅ `>>> int('1000000', 2)`

Output: `64`

- Interprets `'1000000'` as **binary** → only the 7th bit is set → `2⁶ = 64`.

🔍 Summary of Why `9999999999 * 2.1` Might Seem Different

- It's handled just fine, but because the result is a **float**, it has **limited precision**.
- You won't see exact rounding issues at this scale, but they do exist with floats.
- If precise large-number arithmetic is needed with decimals, use Python's `decimal` module.