In Python, **typecasting** (or type conversion) works differently than in Java or C++. Python is dynamically typed, so you don’t explicitly “cast” types — instead, you use **built-in constructor functions** to convert between types.

**Legal Typecasting in Python (works if conversion is meaningful)**

1. **Numbers**
   * int(3.7) → 3 (float → int, truncates decimals)
   * float(5) → 5.0 (int → float)
   * complex(5) → (5+0j) (int → complex)
2. **Strings to Numbers (if the string is numeric)**
   * int("10") → 10
   * float("3.14") → 3.14
   * complex("2+3j") → (2+3j)
3. **Numbers to Strings**
   * str(123) → "123"
   * str(3.14) → "3.14"
4. **Between Collections (when data is compatible)**
   * list((1,2,3)) → [1,2,3] (tuple → list)
   * tuple([1,2,3]) → (1,2,3) (list → tuple)
   * set([1,2,2,3]) → {1,2,3} (list → set)
   * dict([(1,"a"), (2,"b")]) → {1:"a", 2:"b"} (list of tuples → dict)

**Illegal / Not Allowed Typecasting (raises error)**

1. **Nonsensical string to number**
2. int("hello") # ValueError
3. float("abc") # ValueError
4. **Direct conversion between unrelated collections**
5. int([1,2,3]) # TypeError
6. float({1,2,3}) # TypeError
7. **String to dict (unless in valid format)**
8. dict("abc") # TypeError
9. **Custom objects**
   * You can’t automatically cast your class into int/float unless you define \_\_int\_\_, \_\_float\_\_, \_\_str\_\_, etc.

In short:

* **Legal**: If Python has a clear constructor (int(), float(), str(), list(), tuple(), set(), dict()) and the input is compatible.
* **Illegal**: If the conversion doesn’t make sense (e.g., "hello" → int, list → int).