OPERATORS

1. **Major categories of operators in Python:**

- Arithmetic operators: +, -, *, /, //, %, **
- Comparison (Relational) operators: ==, !=, >, <, >=, <=
- Logical operators: and, or, not
- Assignment operators: =, +=, -=, *=, /=, etc.
- Identity operators: is, is not
- Membership operators: in, not in

---

2. **Difference between == and is:**

- == checks for value equality (whether two objects have the same value).
- is checks for identity equality (whether two variables refer to the exact same object in memory).

Example:

```
a = [1, 2, 3]
b = a
c = [1, 2, 3]

print(a == b)
print(a is b)
print(a == c)
print(a is c)
```

3. Operator precedence in Python:

- Determines the order in which operations are evaluated.
- Example: ** (exponentiation) has higher precedence than * (multiplication).
- Use parentheses () to override precedence.

Full precedence table (highest to lowest):

- Parentheses ()
- Exponentiation **
- +x, -x, ~x (Unary operators)
- *, /, //, %
- +, -
- Bitwise shifts <<, >>
- Bitwise AND &
- Bitwise XOR ^
- Bitwise OR |
- Comparison operators (==, !=, <, >, <=, >=)
- Identity operators (is, is not)
- Membership operators (in, not in)
- Logical NOT not
- Logical AND and
- Logical OR or

---

4. **Difference between / and //:**

- / performs floating-point division (returns a float).
- // performs floor division (returns an integer, rounding down).

Example:

```
print(5 / 2)   # 2.5
```

print(5 // 2) *# 2*

---

5. **Purpose of the ** operator:**

- Exponentiation (raises the left operand to the power of the right operand).
- Example: 2 ** 3 returns 8 (2 raised to the power of 3).

---

6. **Logical operators in Python:**

- and (True if both operands are True)
- or (True if at least one operand is True)
- not (Inverts the boolean value)

---

7. **What does the not operator do?**

- Returns True if the operand is False, and vice versa.
- Example: not True → False, not False → True.

---

8. **Difference between identity and equality operators:**

- Identity (is**,** is not): Checks if two variables reference the same object.
- Equality (==**,** !=): Checks if two objects have the same value.

---

9. **What do** in **and** not in **operators do?**

- Check for membership in a sequence (e.g., list, string, tuple, dictionary keys).

- in → True if the value is found.
- not in → True if the value is not found.

Example:

```
nums = [1, 2, 3]
print(2 in nums)      # True
print(5 not in nums)  # True
```

---

10. Comparison operators in conditional statements:

- Used to compare values and return True or False.
- Example:

```
if x > 10:
   print("x is greater than 10")
```

---

11. Chained comparisons (e.g., 10 < x < 20):

- Python allows chaining comparisons for readability.
- Evaluated as (10 < x) and (x < 20).

Example:

```
x = 15
print(10 < x < 20)  # True
```

---

12. Use of += and -= operators:

- Shorthand for augmented assignment (modify and reassign in place).
- Example:

```
x = 5
```

x += 3

x -= 2

---

13. When to prefer is over ==:

- Use is when checking for None, True, False, or object identity.
- Example:

if x is None:

print("x is None")

---

14. Comparing lists with == and is:

- == checks if lists have the same elements in the same order.
- is checks if they are the same object in memory.

Example:

list1 = [1, 2, 3]

list2 = [1, 2, 3]

list3 = list1

print(list1 == list2)

print(list1 is list2)

print(list1 is list3)

---

15. Logical operators with non-boolean values:

- Python treats non-zero numbers and non-empty sequences as True in boolean contexts.
- and returns the last evaluated truth or the first false value.
- or returns the first truth or the last false value.

- not returns True or False based on the truthiness of the operand.

Examples:

```
print(3 and 5)

print(0 and 5)

print(3 or 5)

print(0 or 5)

print(not 3)

print(not "")

print("" or 0)    # Output: 0
```

---

1. print(3 and 5) → Output: 5

How and works:

- Evaluates left to right.
- If all values are truth, returns the last value.
- If any value is false, returns the first false value.

Evaluation:

- 3 is truth (non-zero), so it checks the next value.
- 5 is also truth.
- Since all are truth, returns the last value 5.

2. print(0 and 5) → Output: 0

Evaluation:

- 0 is false (because 0 is false in Python).
- and short-circuits and immediately returns the first false value 0.
- 5 is never evaluated.

3. print(3 or 5) → Output: 3

How or works:

- Evaluates left to right.
- Returns the first truth value.
- If all are false, returns the last false value.

Evaluation:

- 3 is truth, so or short-circuits and returns 3 immediately.
- 5 is never evaluated.

4. print(0 or 5) → Output: 5

Evaluation:

- 0 is false, so or checks the next value.
- 5 is truth, so it returns 5.

5. print(not 3) → Output: False

How not works:

- Converts a value to its **boolean opposite**.
- Always returns True or False.

Evaluation:

- 3 is truth (non-zero), so not 3 → False.

6. print(not "") → Output: True

Evaluation:

- "" (empty string) is false.
- not "" → True (because not inverts false to True).