Operator	Equivalent to
%=	x = x % y

Example Program:

Python
```
counter = 5
counter += 3  # Same as counter = counter + 3
print(f"After += 3: {counter}")  # 8

price = 100
price *= 0.8  # Same as price = price * 0.8 (20% discount)
print(f"After *= 0.8: {price}")  # 80.0
```

Logical Operators: and, or, not

Used to combine conditional statements. They also return a **Boolean** value.

Operator	Description
and	Returns True if **both** operands are true.
or	Returns True if **at least one** operand is true.
not	**Negates** the Boolean value (Flips True to False and vice versa).

Example Program:

Python
```
sunny = True
warm = False

# and
can_swim = sunny and warm # True and False -> False
print(f"Can swim (sunny and warm): {can_swim}")

# or
can_go_outside = sunny or warm # True or False -> True
print(f"Can go outside (sunny or warm): {can_go_outside}")

# not
not_warm = not warm # not False -> True
print(f"Is it not warm: {not_warm}")
```

Identity Operators: is, is not (Comparing memory location/object identity)

- **is:** Returns True if two variables point to the **same object** in memory (i.e., their id() is the same).
- **is not:** Returns True if two variables do **not** point to the same object.

Note: This is different from == which checks if the *values* are equal.

Example Program:

Python
```
list_a = [1, 2, 3]
list_b = [1, 2, 3]
list_c = list_a  # c and a now reference the exact same list object

print(f"list_a == list_b: {list_a == list_b}") # True (Values are equal)
print(f"list_a is list_b: {list_a is list_b}") # False (Different objects/memory IDs)

print(f"list_a == list_c: {list_a == list_c}") # True
print(f"list_a is list_c: {list_a is list_c}") # True (Same object/memory ID)
```

Membership Operators: in, not in (Checking presence in a sequence)

- **in:** Returns True if a value is present in a sequence (like a string, list, or tuple).
- **not in:** Returns True if a value is **not** present in a sequence.

Example Program:

Python
```
my_text = "Hello Python"
vowels = ['a', 'e', 'i', 'o', 'u']

print(f"'P' in my_text: {'P' in my_text}")        # True
print(f"'z' not in my_text: {'z' not in my_text}")  # True
print(f"'e' in vowels: {'e' in vowels}")          # True
```

Operator Precedence and Associativity

Precedence determines the order in which operators are evaluated (e.g., multiplication before addition, like in $2 + 3 * 4$). **Associativity** defines the order of evaluation for operators with the same precedence (usually left-to-right, except for exponentiation **, which is right-to-left).

Order of Precedence (Highest to Lowest, simplified):

1. **Parentheses** ()
2. **Exponentiation** **
3. **Unary operators** (+, -, ~)
4. **Multiplication, Division, Modulus, Floor Division** (*, /, //, %)
5. **Addition and Subtraction** (+, -)
6. **Comparison operators** (==, !=, <, >, etc.)
7. **Identity/Membership operators** (is, in)

8. **Logical operators** (not, and, or)

Example Program:

Python
```
# Standard Math: 10 + (2 * 5) = 20
result_1 = 10 + 2 * 5
print(f"Result 1 (10 + 2 * 5): {result_1}")  # 20 (Multiplication before Addition)

# Use Parentheses to override: (10 + 2) * 5 = 60
result_2 = (10 + 2) * 5
print(f"Result 2 ((10 + 2) * 5): {result_2}") # 60
```

2.4 Input and Output

Using the print() function

The print() function displays output to the console.

Formatting Output, sep, end

- **sep (separator):** Specifies how to separate the arguments passed to print(). Default is a single space.
- **end:** Specifies what to print at the end of the output. Default is a newline character (\n).

Example Program:

Python
```
print("Hello", "World", "!", sep="---")
# Output: Hello---World---!

print("The first line.", end=" ")
print("The second line is concatenated.")
# Output: The first line. The second line is concatenated.

print("List of items:")
for item in ['A', 'B', 'C']:
    print(item, end=", ")
# Output: List of items: A, B, C,
```

Using the input() function to get user input

The input() function pauses the program and waits for the user to type something and press Enter. **It always returns the input as a string.**

Example Program:

Python
```
user_name = input("Please enter your name: ")
age_str = input("Please enter your age: ")

# Input returns a string, so we must convert it for calculations
```

```
user_age = int(age_str)

print(f"Hello, {user_name}! You will be {user_age + 1} next year.")
```

String Formatting: % operator, .format() method, f-strings

These methods allow you to embed variable values into a string elegantly.

1. **% operator (Old style, C-like):**

   Python

   ```
   item = "Laptop"
   price = 999.50
   print("The price of the %s is $%.2f." % (item, price))
   ```

2. **.format() method (Newer style):**

   Python

   ```
   item = "Keyboard"
   price = 75.00
   print("The price of the {} is ${:.2f}.".format(item, price))

   # You can use index or names:
   print("Name: {1}, ID: {0}".format(101, "Dave"))
   ```

3. f-strings (Formatted String Literals - Best practice in modern Python):

   Prepend the string with an f and embed expressions directly inside curly braces {}.

   Example Program (f-strings):

   Python

   ```
   product = "Monitor"
   discount = 0.15
   original_price = 450.00
   final_price = original_price * (1 - discount)

   print(f"Product: {product}")
   # Formatting for currency ($ and 2 decimal places)
   print(f"Original Price: ${original_price:.2f}")
   # Can perform calculations inside {}
   print(f"Discounted Price: ${final_price:.2f}")
   # Conditional formatting
   print(f"Status: {'Expensive' if final_price > 300 else 'Affordable'}")
   ```