Python Day- 2

(RECAP OF PREVIOUS DAY)

Keywords and identifiers, Variables and data types, Type conversion Operators in Python (Arithmetic, Logical, etc.), Operator precedence and associativity. Problems on above concepts.

Keywords and Identifiers

Keywords

- Keywords are reserved words in Python that have predefined meanings.
- They cannot be used as variable names, function names, or identifiers.

Examples of Python Keywords:

and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield

---

Identifiers

Identifiers are the names used to identify variables, functions, classes, or other objects.

They must follow these rules:

1. An identifier name may consist of alphabets(A-Z or a-z), digits or underscore.
2. It can not start with a digit.
3. Cannot use reserved keywords.
4. Python is case-sensitive (e.g., `Var` and `var` are different).

Valid and Invalid Identifiers:

```
# Valid identifiers
var1 = 10
_variable = 20
Var = 30
```

```
# Invalid identifiers
1var = 40       # Starts with a digit
class = 50      # Uses a keyword
```

Variables and Data Types

Variables

- Variables are containers for storing data values.
- You don't need to declare the type explicitly in Python; it is dynamically inferred.

Variable Declaration:

```
# Variable declaration and assignment
x = 5
name = "Alice"
pi = 3.14

# Printing variables
print(x)
print(name)
print(pi)
```

Data Types

Python provides several built-in data types:

1. **Numeric**: `int`, `float`, `complex`
2. **Text**: `str`
3. **Boolean**: `bool`
4. **Sequence**: `list`, `tuple`, `range`
5. **Mapping**: `dict`
6. **Set Types**: `set`, `frozenset`
7. **None Type**: `NoneType`

Examples:

```
# Numeric data types
x = 10       # int
y = 3.14     # float
```

```
z = 3 + 4j    # complex

# Text data type
name = "Alice"  # str

# Boolean data type
flag = True

# Sequence data type
my_list = [1, 2, 3]  # list
my_tuple = (1, 2, 3) # tuple
my_range = range(5)  # range

# Mapping data type
details = {"name": "Alice", "age": 25}  # dict

# Set types
my_set = {1, 2, 3}           # set
my_frozenset = frozenset([1, 2, 3])  # frozenset

# None type
nothing = None
```

Type Conversion

Type conversion is the process of converting one data type into another.

Implicit Type Conversion

Python automatically converts smaller data types to larger ones (e.g., `int` to `float`).

Example:

```
x = 5   # int
y = 2.5 # float
z = x + y
print(z)  # Result is a float: 7.5
```

Explicit Type Conversion

Use built-in functions like `int()`, `float()`, `str()`, etc., to explicitly convert data types.

Example:

```
x = "10"
y = int(x) + 5
print(y)  # Output: 15
```

Operators in Python

Types of Operators

1. **Arithmetic Operators**
2. **Comparison (Relational) Operators**
3. **Logical Operators**
4. **Bitwise Operators**
5. **Assignment Operators**
6. **Special Operators** (Identity operator and Membership operator)

---

1. Arithmetic Operators

Used for mathematical operations.

Operator	Description	Example
+	Addition	`5 + 3 = 8`
-	Subtraction	`5 - 3 = 2`
*	Multiplication	`5 * 3 = 15`
/	Division	`5 / 2 = 2.5`
%	Modulus (Remainder)	`5 % 2 = 1`
//	Floor Division	`5 // 2 = 2`
**	Exponentiation	`5 ** 2 = 25`

Example:

```
x = 10
y = 3
```

```
print(x + y)  # Addition
print(x - y)  # Subtraction
print(x * y)  # Multiplication
print(x / y)  # Division
print(x % y)  # Modulus
print(x // y) # Floor Division
print(x ** y) # Exponentiation
```

---

2. Comparison (Relational) Operators

Used to compare two values.

Operator	Description	Example
==	Equal to	`5 == 3 = False`
!=	Not equal to	`5 != 3 = True`
>	Greater than	`5 > 3 = True`
<	Less than	`5 < 3 = False`
>=	Greater than or equal to	`5 >= 3 = True`
<=	Less than or equal to	`5 <= 3 = False`

Example:

```
x = 5
y = 3
print(x == y)  # False
print(x != y)  # True
print(x > y)   # True
print(x < y)   # False
print(x >= y)  # True
print(x <= y)  # False
```

---

3. Logical Operators

Used to combine conditional statements.

Operator	Description	Example
and	Returns True if both are True	`True and False = False`
or	Returns True if one is True	`True or False = True`
not	Reverses the logical state	`not True = False`

Example:

```
x = True
y = False
print(x and y)  # False
print(x or y)   # True
print(not x)    # False
```

---

4. Bitwise Operators

Used to perform operations on binary numbers.

Operator	Description	Example
&	Bitwise AND	`5 & 3 = 1`
|	Bitwise OR	`5 | 3 = 7`
^	Bitwise XOR	`5 ^ 3 = 6`
~	Bitwise NOT	`~5 = -6`
<<	Bitwise Left Shift	`5 << 1 = 10`
>>	Bitwise Right Shift	`5 >> 1 = 2`

Example:

```
x = 5  # 0101 in binary
y = 3  # 0011 in binary
print(x & y)  # 1 (0001 in binary)
print(x | y)  # 7 (0111 in binary)
print(x ^ y)  # 6 (0110 in binary)
print(~x)     # -6 (inverts all bits)
```

The bitwise AND (`&`) operator compares each bit of the two numbers and returns `1` if both bits are `1`; otherwise, it returns `0`.

Bit position	5 (`0101`)	3 (`0011`)	Result (`&`)
1 (leftmost)	0	0	0
2	1	0	0
3	0	1	0
4 (rightmost)	1	1	1

print(x << 1) # 10 (shifts bits to the left)
print(x >> 1) # 2 (shifts bits to the right)

Another Example:

Operator	Let a = (92)10 =(0101 1100)2 and b = (14)10 = (0000 1110)2	
&	c = (a & b) = 92 & 14	0101 1100 & 0000 1110 ---------- 0000 1100 = $(12)_{10}$
I	c = (a | b) = 92 | 14	0101 1100 | 0000 1110 ---------- 0101 1110 = $(94)_{10}$
^	c = (a ^ b) = 92 | 14	0101 1100 ^ 0000 1110 ---------- 0101 0010 = $(82)_{10}$
~	c = ~a = ~92	c = ~(0101 1100) = 1010 0011
<<	c = a<< 1 = 92<< 1	C = 0101 1100 << 1 = 1011 1000 = $(184)_{10}$

>>	c = a >> 2 = 92>> 2	C = 0101 1100 >> 2 = 0001 0111 = $(23)_{10}$

---

5. Assignment Operators

Used to assign values to variables.

Operator	Description	Example
=	Assign	`x = 5`
+=	Add and assign	`x += 3` (x = 8)
-=	Subtract and assign	`x -= 3` (x = 2)
*=	Multiply and assign	`x *= 3` (x = 15)
/=	Divide and assign	`x /= 3` (x = 1.67)

x += 3 is same as x = x + 3

6. Special Operators

- **identity operators** and **membership operators**. These operators are used for specific purposes such as checking object identity or membership in sequences.

A. Identity Operators

Identity operators are used to compare the **memory location** of two objects to check whether they refer to the **same object**.

Operators

Operator	Description	Example
`is`	Returns `True` if two variables refer to the same object	`x is y`

`is not`	Returns `True` if two variables do not refer to the same object	`x is not y`

Key Notes

- Objects that have the **same value** may not necessarily have the **same memory location**.
- Mutable objects like lists are stored separately in memory even if they have the same values.

Examples

```
# Example 1: Using `is` and `is not`
x = [1, 2, 3]
y = [1, 2, 3]
z = x

print(x is z)        # True (z refers to the same object as x)
print(x is y)  # False (x and y have same value but different objects)
print(x is not y)    # True (x and y are not the same object)


# Example 2: Comparing immutable types
a = 10
b = 10
print(a is b)  # True (Integers with same value share the same memory)
```

B. Membership Operators

Membership operators are used to check whether a value or object is **present in a sequence** (like a string, list, tuple, or dictionary).

Operators

Operator	Description	Example
`in`	Returns `True` if a value is found in the sequence	`'a' in 'apple'`
`not in`	Returns `True` if a value is not found in the sequence	`'x' not in 'apple'`

Examples

 # Example 1: Using `in` and `not in` with strings

```
text = "Hello, world!"
print("Hello" in text)       # True ('Hello' is in the string)
print("Python" not in text)  # True ('Python' is not in the string)
```

 # Example 2: Using `in` and `not in` with lists

```
fruits = ["apple", "banana", "cherry"]
print("apple" in fruits)      # True
print("grape" not in fruits)  # True
```

 # Example 3: Using `in` with dictionaries (checks keys by default)

```
my_dict = {"name": "Alice", "age": 25}
print("name" in my_dict)      # True
print("Alice" in my_dict)     # False (checks keys, not values)
```

Key Differences Between Identity and Membership Operators

Aspect	Identity Operators (`is`, `is not`)	Membership Operators (`in`, `not in`)
Purpose	Compare objects' memory locations	Check for membership in a sequence
Example	`x is y` (checks if `x` and `y` are the same object)	`'a' in 'apple'` (checks if `'a'` is in the string)
Works On	Any objects (numbers, lists, etc.)	Sequences like strings, lists, tuples, dictionaries

---

Quick Recap

- **Identity Operators** check if two objects are the same in memory.

- **Membership Operators** check if an element exists in a sequence.

Operator Precedence and Associativity in Python

In Python, **operator precedence** determines the order in which operators are evaluated in an expression. **Associativity** defines the order in which operators of the same precedence level are evaluated (either left-to-right or right-to-left).

Operator Precedence

Operators in Python have different precedence levels. Higher precedence operators are evaluated first. The following table lists operators in descending order of precedence (from highest to lowest):

Precedence Level	Operator	Description
1 (Highest)	`()`	Parentheses (used for grouping)
2	`**`	Exponentiation
3	`+x`, `-x`, `~x`	Unary operators: positive, negative, bitwise NOT
4	`*`, `/`, `//`, `%`	Multiplication, Division, Floor Division, Modulus
5	`+`, `-`	Addition, Subtraction
6	`<<`, `>>`	Bitwise Shift Operators (Left, Right)
7	`&`	Bitwise AND
8	`^`	Bitwise XOR
9	`	`	`
10	`==`, `!=`, `>`, `<`, `>=`, `<=`, `is`, `is not`, `in`, `not in`	Comparison, Identity, Membership Operators
11	`not`	Logical NOT
12	`and`	Logical AND

13 (Lowest)	`or`	Logical OR

Associativity

When multiple operators with the same precedence appear in an expression, **associativity** determines the order in which they are executed.

Associativity Rules

1. **Left-to-right associativity**: Most operators in Python (e.g., `+`, `-`, `*`, `/`, `//`, `%`, `&`, `|`, etc.) are evaluated from left to right.
2. **Right-to-left associativity**: Some operators, like the exponentiation operator (`**`) and assignment operators (`=`), are evaluated from right to left.

Examples

1. Operator Precedence

```
# Example 1: Exponentiation has higher precedence than multiplication
result = 3 + 2 * 2 ** 2
# Equivalent to: 3 + 2 * (2 ** 2) = 3 + 2 * 4 = 3 + 8 = 11
print(result)  # Output: 11

# Example 2: Parentheses override precedence
result = (3 + 2) * 2 ** 2
# Equivalent to: (3 + 2) * (2 ** 2) = 5 * 4 = 20
print(result)  # Output: 20
```

---

2. Associativity

```
# Example 1: Left-to-right associativity
result = 10 - 5 + 2
# Equivalent to: (10 - 5) + 2 = 5 + 2 = 7
print(result)  # Output: 7

# Example 2: Right-to-left associativity for exponentiation
result = 2 ** 3 ** 2
```

```
# Equivalent to: 2 ** (3 ** 2) = 2 ** 9 = 512
print(result)  # Output: 512
```

Parentheses for Clarity

Using parentheses can make expressions clearer and easier to understand, even when you know the precedence rules.

Example: Clarity with Parentheses

```
# Without parentheses
result = 3 + 5 * 2 - 8 / 4
# Equivalent to: 3 + (5 * 2) - (8 / 4) = 3 + 10 - 2 = 11
print(result)  # Output: 11

# With parentheses for clarity
result = (3 + (5 * 2)) - (8 / 4)
print(result)  # Output: 11
```

Common Mistakes to Avoid

1. Forgetting that `**` (exponentiation) is **right-to-left associative**.

   ```
   print(2 ** 3 ** 2)  # Output: 512 (not 64)
   ```

2. Assuming multiplication (`*`) has higher precedence than exponentiation (`**`).

   ```
   print(2 * 3 ** 2)  # Output: 18 (not 36)
   ```

3. Not using parentheses to group terms when precedence is unclear.

Practice Problems

1. What will be the output of the following expression?

   ```
   result = 10 + 2 * 3 ** 2
   ```

```
print(result)
```

Answer: `28` (Evaluates as `10 + 2 * (3 ** 2)` → `10 + 2 * 9` → `10 + 18`).

2. Rewrite the following expression using parentheses to make the order of operations explicit:

```
result = 4 + 8 / 2 ** 2 * 3
```

Answer: `4 + ((8 / (2 ** 2)) * 3)`

3. Write a program to swap two numbers using bitwise XOR.

```
num1 = 5
num2 = 7
print(f"Before swapping: x = {x}, y = {y}")

# Swapping using XOR
x = x ^ y
y = x ^ y
x = x ^ y

print(f"After swapping: x = {x}, y = {y}")
```

List of programs to practice (Home work)

4. Write a program to find the area of rhombus.
5. Write a program to find the area of parallelogram.
6. Write a program to find the volume and surface area of cube.
7. Write a program to find the volume and surface area of cuboids.
8. Write a program to find the volume and surface area of cylinder.
9. Write a program to find the surface area and volume of a cone.
10. Write a program to find the volume and surface area of sphere.
11. Write a program to find the perimeter of a circle, rectangle and triangle
12. Write a program to Compute Simple Interest.
13. Write a program to Convert Fahrenheit temperature in to Celsius.
14. Write a program to swap the values of two variables.
15. Write a program to swap the values of two variables without using third variable.