Python: Data Types, Operators, and Conditional Statements

Data Types in Python

Data types define the kind of data that variables can hold in Python. Python has dynamically typed variables, meaning you don't need to explicitly declare the type.

Built-in Data Types

1. **Numeric Types**:
   - **int**: Integer values, e.g., `5`, `-10`.
   - **float**: Floating-point numbers, e.g., `3.14`, `-0.01`.
   - **complex**: Complex numbers with a real and imaginary part, e.g., `2 + 3j`.
2. **Sequence Types**:
   - **str**: String, a collection of characters, e.g., `"Hello"`, `'Python'`.
   - **list**: Ordered, mutable collection, e.g., `[1, 2, "Python"]`.
   - **tuple**: Ordered, immutable collection, e.g., `(1, 2, "Python")`.
3. **Mapping Type**:
   - **dict**: Key-value pairs, e.g., `{"name": "Alice", "age": 25}`.
4. **Set Types**:
   - **set**: Unordered, mutable collection of unique elements, e.g., `{1, 2, 3}`.
   - **frozenset**: Immutable version of a set, e.g., `frozenset([1, 2, 3])`.
5. **Boolean Type**:
   - **bool**: Represents `True` or `False`.
6. **Binary Types**:
   - **bytes**: Immutable sequence of bytes.
   - **bytearray**: Mutable sequence of bytes.
   - **memoryview**: Memory view object.
7. **None Type**:

- **NoneType**: Represents the absence of a value, e.g., `None`.

Operators in Python

Operators are symbols or keywords that perform operations on operands.

Types of Operators

1. **Arithmetic Operators**: Used for mathematical operations.
   - `+` (Addition): `5 + 3 = 8`
   - `-` (Subtraction): `5 - 3 = 2`
   - `*` (Multiplication): `5 * 3 = 15`
   - `/` (Division): `5 / 2 = 2.5`
   - `%` (Modulus): `5 % 2 = 1`
   - `**` (Exponentiation): `2 ** 3 = 8`
   - `//` (Floor Division): `5 // 2 = 2`
2. **Comparison (Relational) Operators**: Compare two values and return `True` or `False`.
   - `==` (Equal): `5 == 5` → `True`
   - `!=` (Not Equal): `5 != 3` → `True`
   - `>` (Greater than): `5 > 3` → `True`
   - `<` (Less than): `5 < 3` → `False`
   - `>=` (Greater than or equal to): `5 >= 5` → `True`
   - `<=` (Less than or equal to): `3 <= 5` → `True`
3. **Logical Operators**: Used to combine conditional statements.
   - `and`: `True and False` → `False`
   - `or`: `True or False` → `True`
   - `not`: `not True` → `False`
4. **Assignment Operators**: Assign values to variables.
   - `=`: Assign, e.g., `x = 5`
   - `+=`: Add and assign, e.g., `x += 3` → `x = x + 3`

- `-=`: Subtract and assign, e.g., `x -= 2`
- `*=`: Multiply and assign, e.g., `x *= 2`
- `/=`: Divide and assign, e.g., `x /= 2`

5. **Bitwise Operators**: Operate on binary numbers.
   - `&` (AND): `0b110 & 0b101` → `0b100`
   - `|` (OR): `0b110 | 0b101` → `0b111`
   - `^` (XOR): `0b110 ^ 0b101` → `0b011`
   - `~` (NOT): `~0b110` → `-0b111`
   - `<<` (Left Shift): `0b0011 << 2` → `0b1100`
   - `>>` (Right Shift): `0b1100 >> 2` → `0b0011`
6. **Membership Operators**: Check if a value is in a sequence.
   - `in`: `3 in [1, 2, 3]` → `True`
   - `not in`: `4 not in [1, 2, 3]` → `True`
7. **Identity Operators**: Compare memory locations of objects.
   - `is`: `a is b` → `True` if `a` and `b` reference the same object.
   - `is not`: `a is not b` → `True` if they reference different objects.
8. **Special Operators**:
   - `lambda`: Anonymous function declaration, e.g., `lambda x: x + 10`.

Conditional Statements in Python

Conditional statements allow the execution of specific code blocks based on conditions.

Types of Conditional Statements

1. **if Statement**: Executes a block of code if the condition is `True`.

```
if condition:
    # Code block
```

Example:

```
age = 20
if age >= 18:
    print("You are an adult.")
```

2. **if-else Statement**: Executes one block of code if the condition is `True`, otherwise executes another block.

```
if condition:
    # Code block for True
else:
    # Code block for False
```

Example:

```
age = 16
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

3. **if-elif-else Statement**: Checks multiple conditions sequentially. Executes the first `True` block.

```
if condition1:
    # Code block for condition1
elif condition2:
    # Code block for condition2
else:
    # Code block if none are True
```

Example:

```
marks = 85
if marks >= 90:
    print("Grade: A")
elif marks >= 75:
    print("Grade: B")
else:
    print("Grade: C")
```

4. **Nested if Statements**: Place an `if` statement inside another `if` statement.

```
num = 10
if num > 0:
    if num % 2 == 0:
        print("Positive even number")
    else:
        print("Positive odd number")
```

5. **One-liner Conditional (Ternary Operator)**: Compact syntax for `if-else`.

```
value = x if condition else y
```

Example:

```
age = 20
status = "Adult" if age >= 18 else "Minor"
print(status)
```

Key Points to Remember

- Python supports multiple data types, and variables don't require explicit type declarations.
- Operators are used for computations, comparisons, and logical operations.
- Conditional statements enable decision-making and flow control in programs.
- Proper indentation is critical in Python, as it defines blocks of code.