**Day 4: Python Operators**

Operators are used to testing conditions and manipulating values. Most statements contain expressions: an example of an expression is: 2 + 3. When two objects of a different type, like str and int, are compared they are never equal, except for different numeric types like int and float which can be equal. The <, <=, > and >= operators raise a Type Error exception when any operand is a complex number if the objects are different types that cannot be compared to one another, or where there is no defined ordering**.**

Operators provide the functionality to an expression and can be represented by symbols such as + or by keywords such as and. Operators require data to operate, and this data is known as an operand. In the example: 2 and 3 functions as operands for the + operator, and a and b function as the operands for the / (divide) operator.

|  |  |
| --- | --- |
| **Operation** | **Description** |
| a + b | Sum of a and b |
| a - b | Difference between a and b |
| a \* b | Product of a and b |
| a / b | Quotient of a and b |
| a // b | Floored quotient of a and b. Returns an integer value that was implicitly converted from a floating point value. For example, 8 / 5 = 1.6, when implicitly converted all digits fall away after the dot (.), 8 // 5 = 1 |
| a % b | Remainder of a / b (**modular** operator) |
| -a | a negated |
| +a | a unchanged |

|  |  |
| --- | --- |
| Operation | Description |
| abs(a) | Absolute value or magnitude of a |
| int(a) | a converted to integer |
| float(a) | a converted to floating point |
| complex(re, imag) | A complex number with real part re, imaginary part imag. imag defaults to zero if not specified. |
| divmod(a, b) | The pair (a // b, a % b) |
| pow(a, b) | a to the power b |
| a \*\* b | a to the power b |

|  |  |
| --- | --- |
| **Functions** | **Description** |
| math.trunc(a) | a truncated to Integral |
| round(a | a rounded to the nearest integer |
| math.floor(a) | The greatest integral float <= a |
| math.ceil(a) | The least integral float |

Certainly, here are some key notes on Python operators:

1. Arithmetic Operators:

- Arithmetic operators in Python (+, -, \*, /, //, %, \*\*) are used for performing mathematical operations.

- `/` performs true division, while `//` performs floor division.

- `%` calculates the remainder of a division.

- `\*\*` is used for exponentiation.

2. Comparison Operators:

- Comparison operators (==, !=, <, >, <=, >=) are used to compare values or variables.

- They return a Boolean (True or False) based on the comparison.

3. Logical Operators:

- Logical operators (and, or, not) are used for combining and negating conditions.

- `and` returns True if both conditions are True.

- `or` returns True if at least one condition is True.

- `not` negates a condition.

4. Assignment Operators:

- Assignment operators (=, +=, -=, \*=, /=, //=, %=, \*\*=) are used to assign values to variables.

- Combined assignment operators (e.g., `+=`) perform an operation and assign the result to the variable.

5. Bitwise Operators:

- Bitwise operators (&, |, ^, ~, <<, >>) are used for manipulating individual bits in binary numbers.

- They are often used for low-level operations and bitwise logic.

6. Membership Operators:

- Membership operators (in, not in) are used to test if a value or variable is a member of a sequence (e.g., list, tuple, string).

- They return a Boolean value.

7. \*\*Identity Operators\*\*:

- Identity operators (is, is not) are used to compare the memory addresses of two objects.

- `is` returns True if the objects are the same, and `is not` returns True if they are not the same.

8. Precedence and Associativity:

- Operators have precedence levels, and expressions are evaluated based on these levels.

- Parentheses can be used to specify the order of evaluation.

- Some operators have left-to-right or right-to-left associativity.

9. Expression Evaluation:

- Expressions with operators are evaluated according to operator precedence and associativity.

- The result of an expression can be a value, variable, or Boolean.

10. Logical Short-Circuiting:

- Logical operators (`and` and `or`) in Python use short-circuit evaluation, meaning they may not evaluate all operands if the result can be determined from the first operand.

11. Use in Conditionals:

- Operators are commonly used in conditional statements (if, while) to control the flow of a program based on certain conditions.

12. Use in Loops:

- Operators are used in loops to control iterations and manage loop exit conditions.

13. Custom Operators:

- Python allows the creation of custom operators through operator overloading by defining special methods in classes.

Understanding and using operators is fundamental to programming in Python as they enable you to perform a wide range of operations and comparisons in your code.v