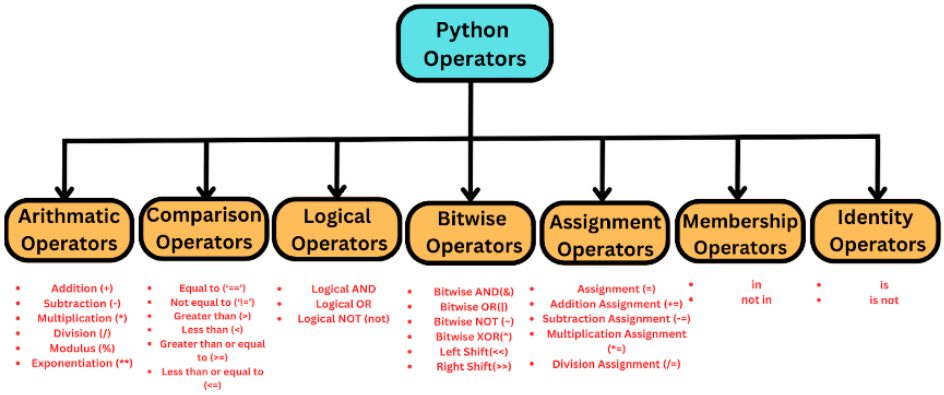
Lesson Plan:

Python Operators

PW SKILLS

Topics to be covered:

1. Python Operators
2. Operators Types
3. Operators Precedence
4. Operators associativity



1. Python Operators:

- Python operators are symbols or special keywords that are used to perform operations on values or variables.

- In Python programming, Operators allow you to manage data, do computations, and make decisions.

2. Operators Types:

There are several types of python operators as follows:

2.1. Arithmetic Operators: Arithmetic operators are used for mathematical operations.

Addition (+): Combining two or more values.

```python
add  = 10 + 5   # add is now 15
```

Subtraction (-): Finding the difference between two values.

```python
sub = 20 - 8  # sub is now 12
```

Multiplication (*): Multiplying values.

```python
mult = 3 * 4  # mult is now 12
```

Division (/): Dividing values.

```python
divide = 24 / 3  # PhysicsWallah is now 8.0
```

Modulus (%): Finding the remainder of a division.

```python
remainder = 17 % 4  # remainder is 1
```

Exponentiation (**): Raising a value to a power.

```python
squared = 5 ** 2  # squared is 25
```

2.2. Comparison Operators:

Comparison operators are used to compare two values and return a Boolean result (True or False).

Equal to ('=='): Checks if two values are equal.

```python
result = 2 == 2  # result is True
```

Not equal to ('!='): Checks if two values are equal.

```python


result = 3 != 2  # result is True
```

Greater than (>): Checks if one value is greater than another.

```python
result = 10 > 9  # result is True
```

Less than (<): Checks if one value is less than another.

```python
result = 10 < 9  # result is False
```

Greater than or equal to (>=): Checks if one value is greater than or equal to another.

```python
result = PhysicsWallah >= PWskills # result is True
```

Less than or equal to (<=): Checks if one value is less than or equal to another.

```python
result = PhysicsWallah <= PWskills # result is False
```

2.3. Logical Operators: Logical operators are used to combine and manipulate Boolean values.

Logical AND (and): Returns True if both conditions are True.

```python
result = (7 > 5) and (8 < 15)  # result is True
```

Logical OR (or): Returns True if at least one condition is True

```python
result = (7 > 5) or (20 < 15)  # result is True
```

Logical NOT (not): Negates a condition (True becomes False, and False becomes True).

```python
result = not (PWskills == Course)  # result is True
```

2.4. Bitwise Operators: In Python, bitwise operators are used to perform operations at the bit-level, manipulating individual bits within integers.

Bitwise AND (&): Returns 1 for each bit position where both operands have a 1.

AND Operation Table

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Example:

```python
a = 5   # 101 in binary
b = 3   # 011 in binary

result = a & b # Bitwise AND

# Binary result: 001, which is 1 in decimal
print("a & b =", result)

#output: a & b = 1
```

Bitwise OR (|): Returns 1 for each bit position where at least one of the operands has a 1.

OR Operation Table

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Example:

```python
a = 5 # 101 in binary
b = 3 # 011 in binary

result = a | b # Bitwise OR
# Binary result: 111, which is 7 in decimal
print("a | b =", result)

#output: a | b = 7
```

Bitwise NOT (~): Inverts the bits; 1s become 0s and vice versa.

NOT Operation Table

a	~a
0	1
1	0

Example:

```python

a = 5 # 101 in binary

result = ~a # Bitwise NOT
# Binary result: 11111010, which is -6 in two's complement
print("~a =", result)

output: ~a = -6
```

Bitwise XOR (^): Returns 1 for each bit position where exactly one of the operands has a 1.

XOR Operation Table

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Example:

```python
a = 5 # 101 in binary
b = 3 # 011 in binary

result = a ^ b # Bitwise XOR

# Binary result: 110, which is 6 in decimal
print("a ^ b =", result)

output: a ^ b = 6
```

Left Shift (<<): Shifts the bits to the left by a specified number of positions, filling with 0s on the right.

```python
a = 5 # 101 in binary

result = a << 2 # Left shift by 2 positions

# Binary result: 10100, which is 20 in decimal
print("a << 2 =", result)

output: a << 2 = 20
```

Right Shift (>>): Shifts the bits to the right by a specified number of positions, filling with 0s on the left (for non-negative numbers) or sign bits (for negative numbers).

```python
a = 16 # 10000 in binary

result = a >> 2 # Right shift by 2 positions
# Binary result: 100, which is 4 in decimal
print("a >> 2 =", result)

output: a >> 2 = 4
```

2.5. Assignment Operators: Assignment operators are used to assign values to variables.

Assignment (=): Assigns a value to a variable.

```python
course = 20
```

Addition Assignment (+=): Adds a value to the variable and updates it.

```python
Course += 5  # Course is now 25
```

Subtraction Assignment (-=): Subtracts a value from the variable and updates it.

```python
Course -= 2  # Course is now 23
```

Multiplication Assignment (*=): Multiplies the variable by a value and updates it.

```python
Course *= 2  # Course is now 46
```

Division Assignment (/=): Divides the variable by a value and updates it.

```python
Course /= 4  # Course is now 11
```

2.6. Membership Operators: Membership operators are used to test if a value is present in a sequence.

in: Checks if a value is present in a sequence (e.g., a string, list, or tuple).

```python
result = "skills" in "PWskills"  # result is True
```

not in: Checks if a value is not present in a sequence.

```python
result = "DataScience" not in "PWskills"  # result is True
```

2.7. Identity Operators: Identity operators are used to compare the memory location of two objects.

is: Returns True if both variables point to the same object.

```python
x = "PWskills"
y = x
result = x is y  # result is True
```

is not: Returns True if both variables point to different objects..

```python
x = "PWskills"
y = "LPU"
result = x is not y  # result is True
```

3. Operator's Precedence:

- Python operators have different levels of precedence, which determine the order in which they are evaluated in an expression. Higher precedence operators are evaluated first.

- Here's a simplified Python operator precedence table, arranged from highest precedence to lowest precedence:

Operators	Meaning
()	Parentheses
**	Exponent
+X, -X, ~X (X is variable)	Unary Plus, Unary Minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor Division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise Shift Operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

1. Parentheses():

Example:

```python
# Example 1: Using parentheses to change order of operations

result = (2 + 3) * 4  # Parentheses ensure addition is done first
print(result)  # Output: 20

# Example 2: Grouping expressions for clarity

total = (10 + 5) + (8 / 2)  # Using parentheses for clarity
print(total)  # Output: 19.0
```

2. Exponentiation' ** ': This operator raises a number to a power.

Example:

```python
# Example 1: Calculating 2 to the power of 3

result = 2 ** 3
print(result)  # Output: 8

# Example 2: Calculating 5 squared
squared = 5 ** 2
print(squared)  # Output: 25
```

3. Unary Plus +X, Unary Minus -X, Bitwise NOT ~X (X is variable):

Example:

```python
# Example 1: Calculating 2 to the power of 3

result = 2 ** 3
print(result)  # Output: 8

# Example 2: Calculating 5 squared
squared = 5 ** 2
print(squared)  # Output: 25
```

4. Multiplication '*', Division '/', Floor Division '//', Modulus '%':

Example:

```python
# Example 1: Multiplication

result = 3 * 4
print(result)  # Output: 12

# Example 2: Division and Modulus

quotient = 15 / 4
remainder = 15 % 4
print(quotient, remainder)  # Output: 3.75 3
```

5. Addition +, Subtraction :

Example:

```python
# Example 1: Addition
sum = 7 + 3
print(sum)  # Output: 10

# Example 2: Subtraction
difference = 12 - 5
print(difference)  # Output: 7
```

6. Bitwise Shift Operators <<, >>:

Example:

```python
# Example 1: Left Shift

result = 8 << 2  # Shift 8 two bits to the left
print(result)  # Output: 32

# Example 2: Right Shift

value = 32
shifted = value >> 2  # Shift 32 two bits to the right
print(shifted)  # Output: 8
```

7. Bitwise AND '&', Bitwise XOR '^', Bitwise OR '|':

Example:

```python
# Declaring variables
a = 10 # Binary representation: 1010
b =6   # Binary representation: 0110

# Example 1: Bitwise AND(&)

result = a & b  # Bitwise AND
print(result)  # Output: 2 Since, 0010 is the result after
applying bitwise AND, its decimal representation is 2

# Example 2: Bitwise XOR(^)

result = a ^ b # Bitwise XOR
print(result)  # Output: 12 Since, 1100 is the result after
applying Bitwise XOR, its decimal representation is 12

# Example 3: Bitwise OR(|)
result = a | b # Bitwise OR
print(result)  # Output: 14 Since, 1110 is the result after
applying Bitwise OR, its decimal representation is 14
```

8. Comparisons, Identity, Membership Operators:

Example:

```python

# Example 1: Comparison Operator (==)

result = 5 == 5
print(result)  # Output: True

# Example 2: Identity Operator (is)

a = [1, 2, 3]
b = a
result = a is b
print(result)  # Output: True

# Example 3: Membership Operator (in)

PWskills = ["DataScience", "DataAnalytics", "DataEngineering"]
result = "DataScience" in PWskills
print(result)  # Output: True
```

9. Logical NOT ‘not’, Logical AND ‘and’, Logical OR ‘or’:

Example:

```python
# Example 1: Logical NOT
result = not True
print(result)  # Output: False

# Example 2: Logical AND
result = True and False
print(result)  # Output: False

# Example 3: Logical OR
result = True or False
print(result)  # Output: True
```

4. Operators Associativity:

Operator associativity in Python is an important concept that determines how operators are grouped and evaluated in expressions. Here's a concise explanation in four bullet points:

1. Order of Evaluation: Operator associativity defines the order in which operators of the same precedence are evaluated within an expression. It specifies whether operators are evaluated from left to right or right to left.

2. Left-to-Right Default: By default, most operators in Python have left-to-right associativity, which means they are evaluated from left to right when they appear in sequence in an expression.

3. Right-to-Left Associativity: Some operators, like the exponentiation operator `**`, have right-to-left associativity, meaning they are evaluated from right to left. This can affect the order of operations in expressions with multiple operators of the same precedence.

4. Control with Parentheses: If you need to override the default associativity and explicitly control the order of evaluation, you can use parentheses `()` to group operations. This allows you to ensure that specific operations are performed before others.

Python Operator's Associativity:

Operator	Associativity	Example	First Evaluation
Parentheses '()'	Left to Right	(3 - 2) * 4	3 - 2
Exponentiation '**'	Right to Left	2 ** 3 ** 2	3 ** 2
Unary Plus +X	Left to Right	+5 + 3	+5
Unary Minus -X	Left to Right	-8 - 2	-8
Bitwise NOT ~X	Left to Right	~10 & 7	~10
Multiplication '*'	Left to Right	2 * 3 * 4	2 * 3
Division '/'	Left to Right	10 / 2 / 5	10 / 2
Floor Division '//'	Left to Right	11 // 3 // 2	11 // 3
Modulus '%'	Left to Right	15 % 4 % 2	15 % 4
Addition '+'	Left to Right	7 + 3 - 1	7 + 3
Subtraction '-	Left to Right	12 - 5 - 2	12 - 5
Bitwise Shift Operators	Left to Right	8 << 2 >> 1	8 << 2
Bitwise AND &	Left to Right	10 & 7 & 5	10 & 7
Bitwise XOR ^	Left to Right	15 ^ 9 ^ 3	15 ^ 9
Bitwise OR |	Left to Right	15 | 9 | 3	15 | 9
Comparison Operators	Left to Right	5 == 5 != 3	5 == 5
Identity Operators	Left to Right	a is b is not c	a is b
Membership Operators	Left to Right	"DataScience" in PWskills	"DataScience" in PWskills
Logical NOT 'not'	Left to Right	not True and False	not True
Logical AND 'and'	Left to Right	True and False or True	True and False
Logical OR 'or'	Left to Right	True or False or True	True or False