DATE : 10 june 2024

DAY : Monday

TOPICS: 1.Python strings and commonly used string methods 2. Operators 3. Operator Precedence

STRING :

In Python, strings are used for representing textual data. A string is a sequence of characters enclosed in either single quotes (") or double quotes (""). The Python language provides various built-in methods and functionalities to work with strings efficiently.

STRING METHODS :

upper() method

Description : Converts the string to uppercase.

```
#str.upper() method
Stu = "diksha"
print(id(Stu))
Stu1 = Stu.upper()
print(Stu1)
print(id(Stu1))
```

```
132016382514992
DIKSHA
132016382411824
```

lower() method

Description : Converts the string to lowercase

```
#str.lower()
Stu = "MUKTA"
print(Stu)
Stu2 = Stu.lower()
print(Stu2)
```

```
MUKTA
mukta
```

capitalize() method

Description: Converts the first character to uppercase and the rest to lowercase.

```
#str.title()
name = "diksha thakur"
y = name.title()
y
```

```
'Diksha Thakur'
```

title() method

Description : Converts the string to title case.

```
#str.capitalize()
school = "dav school"
X = school.capitalize()
X
```

```
'Dav school'
```

swapcase() method

Description : Swaps case, converting lowercase to uppercase and vice versa.

```
#str.swapcase
name = "mUkTawali tOdiwaL"
z = name.swapcase()
```

endswith() method

Description : Returns True if the string ends with the specified suffix.

```
#endswith()
name = "diksha thakur "
name.endswith(' ')
```

```
True
```

center() method

Description : Centers the string in a field of given width.

```
#str.center()
name = "KULDEEP"
print(len(name))
Y = name.center(15)
print(len(Y))
Y
```

```
7
15
'    KULDEEP    '
```

expandtabs() method

In Python, the str.expandtabs() method is used to replace tab characters ('\t') in a string with the appropriate number of spaces, based on a specified tab size. This method is particularly useful when you want to format text with consistent spacing.

The expandtabs() method takes an optional argument, which is the tab size. If no argument is provided, the default tab size is 8 spaces.

```
#str.expandtabs()
original_string = "My\tname\tis\tDiksha\tThakur."
print(original_string)
# Using expandtabs() with default tab size (8 spaces)
expanded_string_default = original_string.expandtabs()
print(expanded_string_default)
# Using expandtabs() with a custom tab size (e.g., 4 spaces)
expanded_string_custom = original_string.expandtabs(4)
print(expanded_string_custom)
```

```
My      name    is      Diksha  Thakur.
My      name    is      Diksha  Thakur.
My  name    is  Diksha  Thakur.
```

isaplha() method

Description : Returns True if all characters are alphabetic

```
#str.isaplha()
My_self = "oamaboy"
My_self.isalpha()
```

```
True
```

isnumeric() method

Description : Returns True if all characters are numeric.

```
#str.isnumeric()
My_self = "Kuldeep"
L = My_self.isnumeric()
print(L)
```

```
you = "1234"
you.isnumeric()
```

```
False
True
```

alphanum() method

Description : Returns True if all characters are alphanumeric

```
#str.alphanum
you = "D1234m"
you.isalnum()
```

```
True
```

islower() method

Description : Returns True if all characters are lowercase.

```
#str.islower
Name = "Diksha"
Name.islower()
```

```
False
```

index() method

Description : Returns the lowest index where the substring is found.

```
Name = "Diksha"
u = Name.index('i')
u
```

```
1
```

find() method

Description : Returns the lowest index where the substring is found.

```
Name = "Diksha"
u = Name.find('h')
u
```

```
4
```

casefold() method

The str.casefold() method in Python is used to perform a case-insensitive string comparison. It returns a new string with all the characters converted to lowercase, but it goes a step further than str.lower(). The casefold() method not only converts the string to lowercase but also performs additional transformations to make the comparison more robust in the context of Unicode characters

```
original_string = "Hello World"
casefolded_string = original_string.casefold()
lowere_string = original_string.lower()
print("Original String:", original_string)
print("Casefolded String:", casefolded_string)
print("Lowered String:", lowere_string)
```

```
Original String: Hello World
Casefolded String: hello world
Lowered String: hello world
```

The casefold() method is particularly useful when you need to perform case-insensitive string comparisons in a way that is robust to different Unicode representations of the same character. The primary difference between str.casefold() and str.lower() is in how they handle certain Unicode characters. casefold() is more aggressive in its approach, making it suitable for case-insensitive comparisons in a broader range of situations, especially when dealing with different language characters and special symbols. Both str.casefold() and str.lower() methods in

Python are used to convert a string to lowercase. However, there are subtle differences between them, primarily related to how they handle certain Unicode characters. Let's delve deeper into these differences with examples.

handling special character

```
s1 = "Straße" # German sharp-s
s2 = "strasse" # Latin letter 's' followed by 't', 'r', 'a', 's', 's', 'e'
# Using str.lower()
lower_s1 = s1.lower()
lower_s2 = s2.lower()
# Using str.casefold()
casefold_s1 = s1.casefold()
casefold_s2 = s2.casefold()
print("Lowercase s1:", lower_s1)
print("Lowercase s2:", lower_s2)
print("Casefolded s1:", casefold_s1)
print("Casefolded s2:", casefold_s2)
```

```
Lowercase s1: straße
Lowercase s2: strasse
Casefolded s1: strasse
Casefolded s2: strasse
```

In this example, str.casefold() not only converts the German sharp-s (ß) to lowercase but also replaces it with 'ss', making the two strings equal. str.lower() does not perform this additional transformation.

handling unicode equivalents

```
s1 = "Mañana" # Spanish 'ñ'
s2 = "manana" # Latin letter 'n' followed by 'a', 'n', 'a', 'n', 'a'
# Using str.lower()
lower_s1 = s1.lower()
lower_s2 = s2.lower()
# Using str.casefold()
casefold_s1 = s1.casefold()
casefold_s2 = s2.casefold()
print("Lowercase s1:", lower_s1)
print("Lowercase s2:", lower_s2)
print("Casefolded s1:", casefold_s1)
print("Casefolded s2:", casefold_s2)
```

```
Lowercase s1: mañana
Lowercase s2: manana
Casefolded s1: mañana
Casefolded s2: manana
```

In this example, str.casefold() handles the Spanish 'ñ' and its lowercase equivalent in a way that makes the two strings equal. str.lower() does not perform this transformation.

Conclusion:

str.lower() is suitable for most case-insensitive comparisons but may not cover all Unicode characters, especially those with special equivalents.

str.casefold() is more aggressive and covers a broader range of Unicode characters, making it suitable for case-insensitive comparisons in diverse scenarios, such as different language characters and special symbols.

In general, if you need a case-insensitive comparison and want to be thorough, especially when dealing with internationalization and Unicode, str.casefold() is often a safer choice.

Practice Questions(String) :

Write a Python program that reverses a given string.

```
#reverse
str = "komal"
print(str[::-1])
```

```
lamok
```

Write a program to check if a string is a palindrome.

```
#palindrome
str = "noon"
rev = str[::-1]
if rev in str:
  print("palindrome")
else:
  print("not palindrome")
```

```
palindrome
```

Implement a function to capitalize the first letter of each word in a sentence.

```
#capitalize the first letter of each word of sentence
str = "my name is Komal"
print(str.title())
```

```
My Name Is Komal
```

Create a formatted string using f-strings that includes variables and their values.

```
#formatting string
age = "20"
print(f" The age is {age}")
```

```
 The age is 20
```

Given a list of names, format them into a bulleted list using string concatenation or join method.

```
#concatenate the names
name_1 = "komal"
name_2 = " neha"
name_3 = " heena"
all_names = name_1+name_2+name_3
print(all_names)
```

```
komal neha heena
```

Write a function to find the index of the first occurrence of a substring in a given string.

```
#find index of substring from string
str = 'banana'
print(str.index('an'))
```

```
1
```

Count the occurrences of a specific word in a paragraph.

```
#count occurence of specific word in a paragraph
str = 'Nature is the connection between the physical world surrounding us and the life inside us. Nature is God’s most precious and valuable
print(str.count('a'))
```

```
22
```

Write a program to extract all email addresses from a given text.

```
#extract email from text
str = 'Email of komal is heerkomal29@gmail.com\nEmail of neha is neha34@gmail.com'
email = '@gmail.com'
if email in str:
  print(str)
else:
  print("Email is not present")
```

```
Email of komal is heerkomal29@gmail.com
Email of neha is neha34@gmail.com
```

Write a program to extract all email addresses from a given text.

```
#extract last three words from string
str = 'elephant'
print(str[5::1])
```

```
ant
```

Concatenate two strings with a space in between.

```
#concatenate 2 strings with space
f_name = 'komal'
l_name = 'heer'
full_name =f_name+' '+l_name
print(full_name)
```

```
komal heer
```

Given a string containing a sentence, use a string method to convert the entire string to uppercase.

```
#upper case
str = 'komal heer'
str.upper()
```

```
'KOMAL HEER'
```

You have a string with extra spaces at the beginning and end. Use a string method to remove these extra spaces.

```
#remove beginning and ending space
str = ' komalheer '
str.strip()
```

```
'komalheer'
```

Given a string, find the first occurrence of the substring "Python" and print its position.

```
#find substring from string
str = 'programming language Python is a high-level, general-purpose programming language. Python is dynamically typed and garbage-collected'
str.index("Python")
```

```
21
```

You have a string with words separated by commas. Use a string method to split the string into a list of words.

```
#split string into list of words
str = 'c, java, swift, css, html'
str.split(",")
```

```
['c', ' java', ' swift', ' css', ' html']
```

Given a string containing a sentence, replace all occurrences of the word "Java" with "Python".

```
#replace Java with python
str = 'Java is a widely-used programming language for coding web applications. Java is a multi-platform and network-centric language that ca
str.replace('Java','Python')
```

```
'Python is a widely-used programming language for coding web applications. Python is a multi-platform and network-centric language that
can be used as a platform in itself'
```

You have a string containing a mix of uppercase and lowercase characters. Use a string method to convert the entire string to lowercase.

```
#lower case
str = 'My Name is KOMAL HEER from CSE 3rd YEAR'
str.lower()
```

```
'my name is komal heer from cse 3rd year'
```

Given a string, check if it starts with the prefix "Hello".

```
#check whether the string is starts with Hello
str = 'Hello my name is komal'
str.startswith('Hello')
```

```
True
```

You have a string with numbers and letters. Use a string method to check if the string is alphanumeric.

```
#check string having alphanumeric value or not
str = 'heerkomal2911'
str.isalnum()
```

```
True
```

Given a string containing multiple words, use a string method to join the words with a hyphen ("-").

```
#join each word of string with hyphen
str = 'C Java Python Swift CSS HTML'
print(str.split(" "))
print('-'.join(str.split()))
```

```
['C', 'Java', 'Python', 'Swift', 'CSS', 'HTML']
C-Java-Python-Swift-CSS-HTML
```

You have a string with different words separated by spaces. Use a string method to count the number of words in the string.

```
str = "k o m a l h e e r"
res = len(str)
print(res)
```

```
17
```

Given a string containing a sentence, use a string method to capitalize the first letter of each word.

```
#capitalize the first letter of each word
str = 'hello how are you'
str.title()
```

```
'Hello How Are You'
```

You have a string containing a URL. Use a string method to check if the URL ends with ".com".

```
#check URL
str = 'http://example.com'
str.endswith(".com")
```

```
True
```

You have a string containing a sentence. Use a string method to find the number of times the word "data" appears in the string.

```
#find number of times word "data" appears in string
str  = 'data are a collection of discrete or continuous values, data can be texts or numbers written on papers, data is information that has
str.count("data")
```

```
3
```

Given a string, use a string method to reverse the string.

```
#reverse the string
str = 'komalheer'
str[::-1]
```

```
'reehlamok'
```

You have a string that contains both letters and digits. Use a string method to extract only the digits from the string.

```
#extract digits from string
str = 'heerkomal2911'
for i in str:
  if(i.isdigit()):
    print(i, end="")
```

```
2911
```

Python Operators :

There are seven kinds of operators in Python.

Arithmetic Operators

Bitwise Operators

Assignment Operators

Comparison Operators /Relational Operators

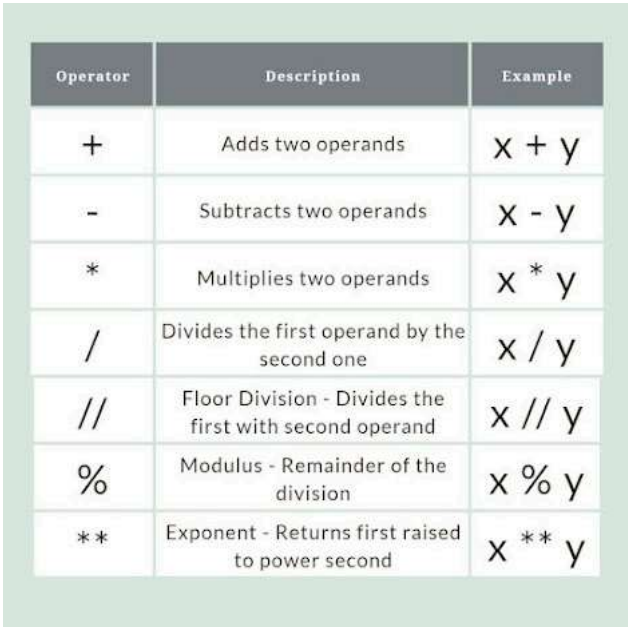
Logical Operators

Identity Operators

Membership Operators

Arithmetic Operators

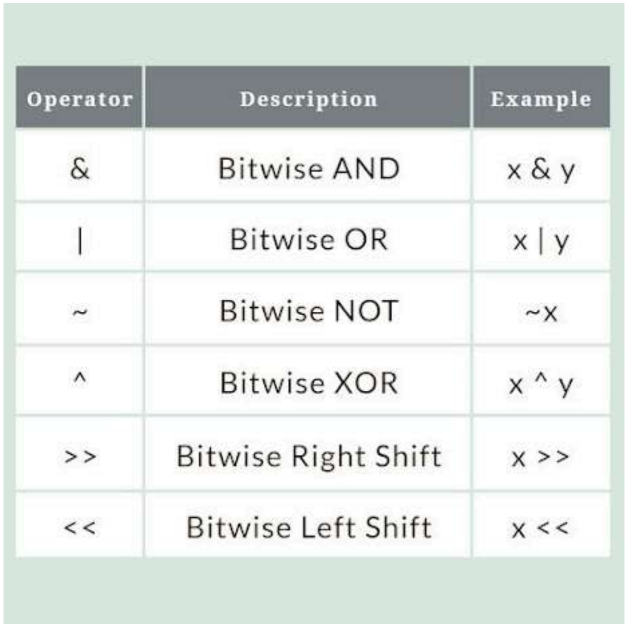
Arithmetic Operators are used to perform basic mathematical arithmetic operators like addition, subtraction, multiplication, etc. The following table lists out all the arithmetic operators in Python.



Operator	Description	Example
+	Adds two operands	x + y
-	Subtracts two operands	x - y
*	Multiplies two operands	x * y
/	Divides the first operand by the second one	x / y
//	Floor Division - Divides the first with second operand	x // y
%	Modulus - Remainder of the division	x % y
**	Exponent - Returns first raised to power second	x ** y

Bitwise Operators

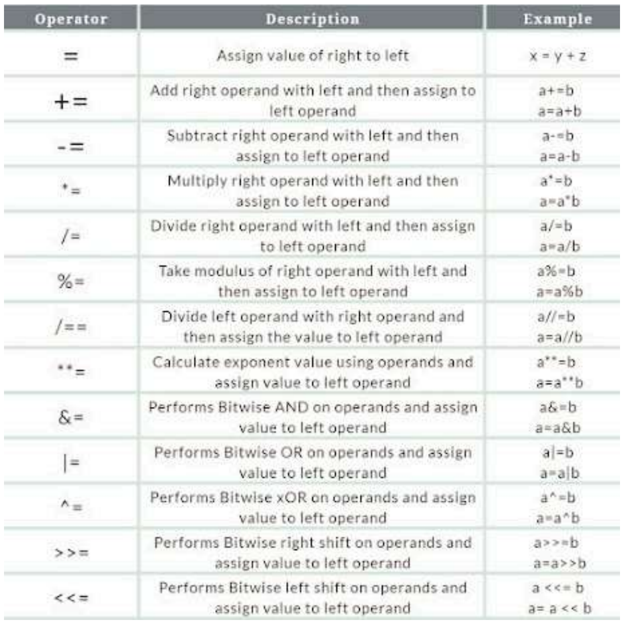
Bitwise Operators are used to perform bit level operations. The following table lists out all the bitwise operators in Python.



Operator	Description	Example
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise Right Shift	x >>
<<	Bitwise Left Shift	x <<

Assignment Operators

Assignment Operators are used to assign or store a specific value in a variable. The following table lists out all the assignment operators in Python.



Operator	Description	Example
=	Assign value of right to left	x = y + z

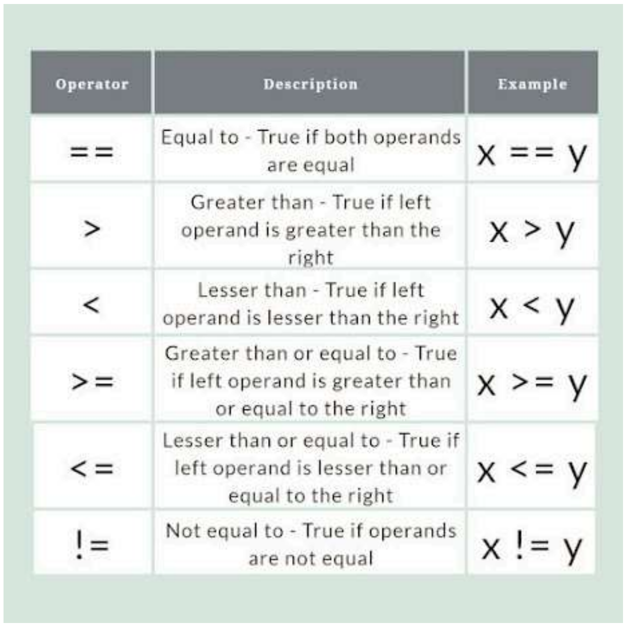
+=	Add right operand with left and then assign to left operand	a+=b
		a=a+b

-=	Subtract right operand with left and then assign to left operand	a-=b
		a=a-b

*=	Multiply right operand with left and then assign to left operand	a*=b a=a*b
/=	Divide right operand with left and then assign to left operand	a/=b a=a/b
%=	Take modulus of right operand with left and then assign to left operand	a%=b a=a%b
/==	Divide left operand with right operand and then assign the value to left operand	a//=b a=a//b
**=	Calculate exponent value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

Comparison Operators

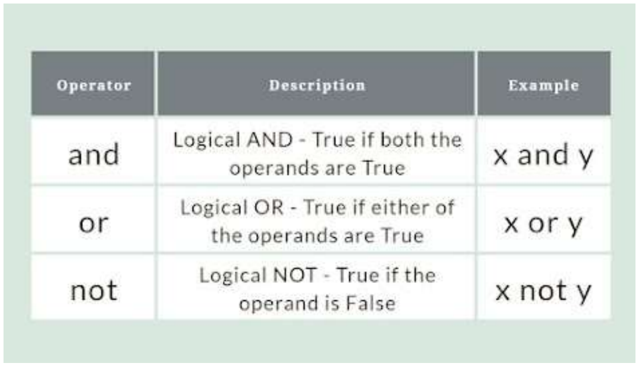
Comparison Operators are used to compare two operands. The following table lists out all the Comparison operators in Python.



Operator	Description	Example
==	Equal to - True if both operands are equal	x == y
>	Greater than - True if left operand is greater than the right	x > y
<	Lesser than - True if left operand is lesser than the right	x < y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Lesser than or equal to - True if left operand is lesser than or equal to the right	x <= y
!=	Not equal to - True if operands are not equal	x != y

Logical Operators

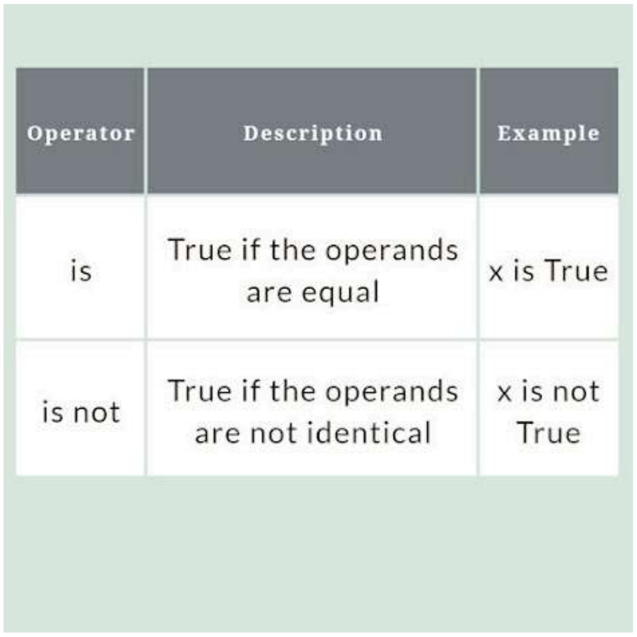
Logical Operators are used to combine simple conditions and form compound conditions. The following table lists out all the Logical operators in Python.



Operator	Description	Example
and	Logical AND - True if both the operands are True	x and y
or	Logical OR - True if either of the operands are True	x or y
not	Logical NOT - True if the operand is False	x not y

Identity Operators

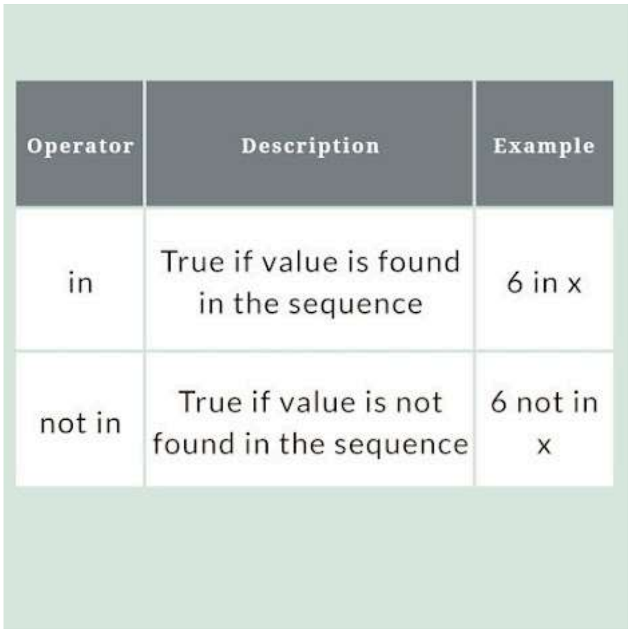
Identity Operators are used to check if two variables point to same reference of an object in Python. The following table lists out the two Identity operators in Python.



Operator	Description	Example
is	True if the operands are equal	x is True
is not	True if the operands are not identical	x is not True

Membership Operators

Membership Operators are used to check if an element or item is present in the given collection or sequence. The following table lists out the two Membership operators in Python.



Operator	Description	Example
in	True if value is found in the sequence	6 in x
not in	True if value is not found in the sequence	6 not in x

Practice Questions (Operators)

1. You have two variables, a and b, containing integer values. Use arithmetic operators to calculate and print the sum, difference, product, and quotient of these variables.

```
a = 9
b = 4
print(f" Addition is {a+b}")
print(f" Subtraction is {a-b}")
print(f" Multiplication is {a*b}")
print(f" Division is {a/b}")
```

```
 Addition is 13
 Subtraction is 5
```

```
Multiplication is 36
Division is 2.25
```

Given a variable x containing an integer value, use the modulus operator to check if x is even or odd.

```
x = 35
if x%2==0:
  print(f"{x} is even")
else:
  print(f"{x} is odd")
```

```
35 is odd
```

You have two variables, a and b. Use comparison operators to check if a is greater than b, and print the result.

```
a = 3
b = 6
if a>b:
  print(f"{a} is greater than {b}")
else:
  print(f"{a} is less than {b}")
```

```
3 is less than 6
```

Given two boolean variables, p and q, use logical operators to evaluate and print the result of p AND q, p OR q, and NOT p.

```
p = True
q = False
print(f"{p} and {q} is {p and q}")
print(f"{p} or {q} is {p or q}")
print(f"not {p} is {not p}")
```

```
True and False is False
True or False is True
not True is False
```

You have a variable n containing an integer value. Use the bitwise AND, OR, and XOR operators to perform operations with another integer m.

```
n = 7
m = 5
print(f"{n} and {m} is {n&m}")
print(f"{n} or {m} is {n|m}")
print(f"{n} xor {m} is {n^m}")
```

```
7 and 5 is 5
7 or 5 is 7
7 xor 5 is 2
```

Given a variable y containing a floating-point number, use the floor division operator to divide y by 2 and print the result.

```
y = 35.3
print(f"{y}//2 is {y//2}")
```

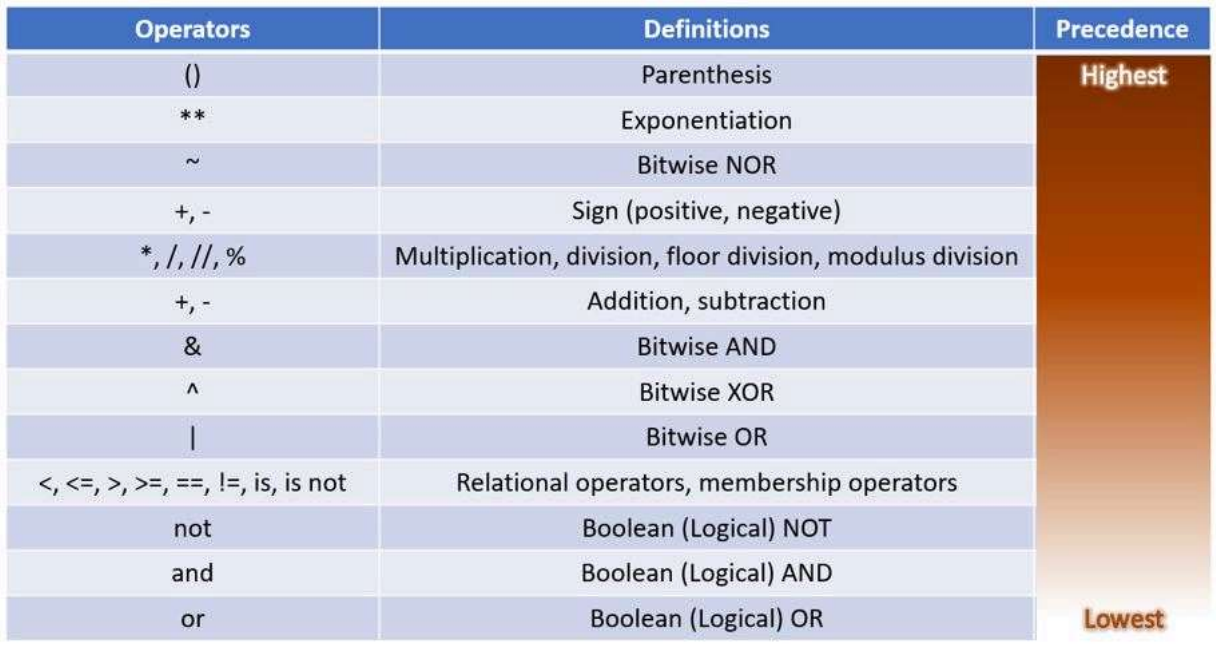
```
35.3//2 is 17.0
```

You have two strings, str1 and str2. Use the concatenation operator to join these strings and print the result.

```
#concatenate two strings
str_1 = 'komal'
str_2 = 'heer'
print(str_1+str_2)
```

```
komalheer
```

Operator Precedence



Operators	Definitions	Precedence
()	Parenthesis	Highest
**	Exponentiation	
~	Bitwise NOR	
+, -	Sign (positive, negative)	
*, /, //, %	Multiplication, division, floor division, modulus division	
+, -	Addition, subtraction	
&	Bitwise AND	
^	Bitwise XOR	
|	Bitwise OR	
<, <=, >, >=, ==, !=, is, is not	Relational operators, membership operators	
not	Boolean (Logical) NOT	
and	Boolean (Logical) AND	
or	Boolean (Logical) OR	Lowest

Practice Questions (Operator Precedence)

You have the expression 5 + 3 * 2. Without using parentheses, calculate the result and then use parentheses to explicitly show the precedence.

```
str = '5 + 3 * 2'
print(str)
ans_1 = 5 + 3 * 2 #without parenthesis
print(ans_1)
ans_2 = 5 + (3 * 2) #with parenthesis
print(ans_2)
```

```
5 + 3 * 2
11
11
```

Given the expression 10 / 2 + 3, evaluate it step by step according to operator precedence rules and then use parentheses to clarify the order of operations.

```
str = '10 / 2 + 3'
print(str)
ans_1 = 10 / 2 + 3 #without parenthesis
print(ans_1)
ans_2 = (10 / 2) + 3 #with parenthesis
print(ans_2)
```

```
10 / 2 + 3
8.0
8.0
```

You have the expression 4 + 5 * (3 - 2). Evaluate it and explain why the result differs from 4 + 5 * 3 - 2.

```
str_1 = '4 + 5 * (3 - 2)'
print(str_1)
ans_1 = 4 + 5 * (3 - 2) #in which firstly the bracket is solve (eg- 3-2)
print(ans_1)
str_2 = '4 + 5 * 3 - 2'
print(str_2)
ans_2 = 4 + 5 * 3 - 2 #it is solve using precedence (eg- 5*3)
print(ans_2)
```

```
4 + 5 * (3 - 2)
9
4 + 5 * 3 - 2
17
```

Consider the expression 3 + 4 * 2 ** 2. Calculate the result by following the operator precedence rules and then rewrite it using parentheses to make the precedence explicit.

```
str = '3 + 4 * 2 ** 2'
print(str)
res_1 = 3 + 4 * 2 ** 2 #without parenthesis
print(res_1)
res_2 = 3 + 4 * (2 ** 2) #with parenthesis
print(res_2)
```

```
3 + 4 * 2 ** 2
19
19
```

Given the expression 10 - 3 + 2 * 4 / 2, evaluate it by following the operator precedence and then rewrite it using parentheses to clarify the operations.

```
str = '10 - 3 + 2 * 4 / 2'
print(str)
res_1 = 10 - 3 + 2 * 4 / 2 #without parenthesis
print(res_1)
res_2 = 10 - 3 + 2 * (4 / 2) #with parenthesis
print(res_2)
```

```
10 - 3 + 2 * 4 / 2
11.0
11.0
```

You have the expression 5 * (2 + 3) ** 2. Calculate the result and explain why parentheses are necessary in this case.

```
str = '5 * (2 + 3) ** 2'
print(str)
res_1 = 5 * (2 + 3) ** 2 #parenthesis is necessary to understood which operation is performed first.
print(res_1)
```

```
5 * (2 + 3) ** 2
125
```

Given the expression 8 / 2 * 3, evaluate it according to operator precedence and then rewrite it using parentheses to show the correct order of operations.

```
str = '8 / 2 * 3'
print(str)
res_1 = 8 / 2 * 3
print(res_1)
res_2 = (8 / 2) * 3
print(res_2)
```

```
8 / 2 * 3
12.0
12.0
```

Consider the expression (6 + 4) / 2 - 3. Evaluate it step by step and explain how the parentheses affect the result