

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))
```

```
⇒ 139091965595184
   DIKSHA
   139091965599536
```

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.
```

isalpha() method

Description : Returns True if all characters are alphabetic

```
#str.isalpha()
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()
lower_string = original_string.lower()
print("Original String:", original_string)
print("Casefolded String:", casefolded_string)
print("Lowered String:", lower_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: straÙe
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 to print the type of the data in variable and its length.

```
name = 'Jasmine'
Name = "Jasmine"
print(name)
print(Name)
print(type(name))
print(len(name))
```

```
⇒ Jasmine
   Jasmine
   <class 'str'>
   7
```

Write a Python program that reverses a given string.

```
#reversing the string
str = "Jasmine"
str[::-1]
```

```
⇒ 'enimsaJ'
```

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

```
#check for a string palindrome
str = "madam"
rev = str[::-1]
print(rev)
if rev in str:
    print("palindrome")
else:
    print("not palindrome")
```

```
→ madam  
palindrome
```

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

```
#capitalise first letter of each word in sentence  
str = "my name is jasmine"  
print(str.title())
```

```
→ My Name Is Jasmine
```

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

```
#string concatenation  
f_name = "Jasmine"  
m_name = " Kaur "  
l_name = "Kharay"  
fu_name = f_name + m_name + l_name  
print(fu_name)
```

```
→ Jasmine Kaur Kharay
```

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

```
# f or formatting string  
str = "jasmine"  
print(f" the string is {str}")
```

```
→ the string is jasmine
```

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

```
#extract the email addresses  
information = "The email of jasmine is abc@gmail.com and email of arsh is pqr@gmail.com"  
email = '@gmail.com'  
if email in information:  
    print(information)  
else:  
    print ("Email is not present")
```

```
→ The email of jasmine is abc@gmail.com and email of arsh is pqr@gmail.com
```

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


```
str = "banana"
print(str.index('an'))
```

➞ 1

Count the occurrences of a specific word in a paragraph.

```
str = "Stop having hard conversations with people who don't want to change. Stop showing up"
str.count('Stop')
```

➞ 4

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

```
str = "My name is Jasmine. "
str[15::]
```

➞ 'ine. '

Concatenate two strings with a space in between.

```
str1 = "Hello"
str2 = "World"
combined_string = str1 + ' ' + str2
print(combined_string)
```

➞ Hello World


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

```
#practice questions
str = "jasmine"
str.upper()
```

➞ 'JASMINE'

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

```
str = " jasmine"
str.strip()#to remove the begining and ending space
```

 `'jasmine'`

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

```
str = "programming language python is a general purpose and high - level language "  
str.index('python')
```

 `21`


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

```
str = "Java is a widely used programming language "  
str.replace('Java','Python')
```

 `'Python is a widely used programming language '`


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

```
#split the string into list of words  
str = "CSS, Java, Python, HTML"  
str.split(",")
```

 `['CSS', ' Java', ' Python', ' HTML']`

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

```
str = "HeLlO wOrld"  
str.lower()
```

 `'hello world'`

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

```
str = "hello world"  
str.startswith("hello")
```

 `True`

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

```
str = "12jasm1ne"  
str.isalnum()
```

⇒ True

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

```
str = "C CSS Java Python"  
print(str.split())  
'-'.join(str.split())
```

⇒ ['C', 'CSS', 'Java', 'Python']
'C-CSS-Java-Python'

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

```
str = "j a s m i n e"  
len(str)
```

⇒ 13

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

```
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'  
str.count("data")
```

⇒ 3

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

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

⇒ 'ruaK enimsaJ'

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 = 'Jasmine0306'  
for i in str:  
    if(i.isdigit()):  
        print(i, end="")
```

⇒ 0306

✓ 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.

<i>Operator</i>	<i>Name</i>	<i>Example</i>
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Bitwise Operators

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

<i>Operator</i>	<i>Name</i>	<i>Description</i>
&	AND	Sets each bit to 1 if both the bits are 1
	OR	Sets each bit to 1 if one of the two bits is 1
^	XOR	Sets each bit to 1 if only one of the two bits is 1
~	NOT	Inverse all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

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	Example	Same as
= Assign value of right side of expression to left side operand	x = 5	x = 5
+= Add AND: Add right-side operand with left side operand and then assign to left operand	x += 5	x = x + 5
-= Subtract AND: Subtract right operand from left operand and then assign to left operand	x -= 3	x = x - 3
*= Multiply AND: Multiply right operand with left operand and then assign to left operand	x *= 5	x = x * 5
/= Divide AND: Divide left operand with right operand and then assign to left operand	x /= 3	x = x / 3
%= Modulus AND: Takes modulus using left and right operands and assign the result to left operand	x %= 3	x = x % 3
//= Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	x //= 3	x = x // 3
**= Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	x **= 3	x = x ** 3
&= Performs Bitwise AND on operands and assign value to left operand	x &= 3	x = x & 3
= Performs Bitwise OR on operands and assign value to left operand	x = 5	x = x 5
^= Performs Bitwise xOR on operands and assign value to left operand	x ^= 3	x = x ^ 3
>>= Performs Bitwise right shift on operands and assign value to left operand	x >>= 3	x = x >> 3
<<= Performs Bitwise left shift on operands and assign value to left operand	x <<= 5	x = x << 5

Comparison Operators

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

<i>Operator</i>	<i>Name</i>	<i>Example</i>
==	Equal	a == b
!=	Not equals	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

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.

<i>Operator</i>	<i>Description</i>	<i>Example</i>
and	Returns True if both conditions are true	a < 5 and a < 10
or	Returns True if one of the conditions is true	a < 6 or a < 5
not	Opposite of the result, returns False if the result is true	not(a < 4 and a < 8)

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.

Description**Example**

is

Returns True if both variables are the same object:
x = 5; y = 5

x is y

is not

Returns True if both variables are not the same object
a = 6; b = 10

a is not

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.

Description**Example**

is

Returns True if both variables are the same object:
x = 5; y = 5

x is y

is not

Returns True if both variables are not the same object
a = 6; b = 10

a is not

✓ 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 = 'Jasmine'
str_2 = 'Kaur'
print(str_1+str_2)
```

➞ JasmineKaur

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
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) *
```

```
⇒ 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.