

Unit 4 - Lesson 1 - Introduction to Strings, String Operations

About this unit

Introduction to Strings, String Operations

Introduction to Strings

Unit • 100% completed



String Operations

Unit • 100% completed

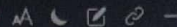


Basics of Strings and String Operations - Python

Assessment

30.1.1. Introduction to Strings

04:22



Strings in python are represented with prefixing and suffixing the characters with quotation marks (either single quotes (' ') or double quotes (" ")).

- Characters in a string are accessed using an **index** which is an integer (either positive or negative).
- It starts from **0 to n-1**, where **n** is the number of characters in a string.
- **Strings** are immutable which means contents **cannot** be changed once they are created.
- The function **input()** in python is a string by default.

Click on the [Live Demo](#) button to know about strings in python.

Write a code snippet that prompts the user to input a string using the `input()` function and print the entered string as shown in the sample test case.

Sample Input and Output:

```
str: Strings in Python are immutable
Strings in Python are immutable
```

Sample Test Cases

Explorer

String1.py

```
1 str1 = str(input("str: "))
2
3 print(str1)
4
```

Terminal

Test cases

String Operations

About this unit

String Operations

? Operations of Strings

Question

? Understanding Slicing Operator

Question

? Concatenation of strings

Question

? Program to remove a character from string based on integer value given by the user.

Question

? Understanding Membership Operator

Question

30.2.1. Operations of Strings

In Python, there are 5 fundamental operations which can be performed on strings:

1. Indexing
2. Slicing
3. Concatenation
4. Repetition
5. Membership

Indexing: To access a specific character from a string, we use its position called **Index** which is enclosed within square brackets `[]`. Index value always starts with `0`. Python has two types of indexing:

1. **Positive Indexing:** It begins from the first character of a string, starting with `0`.
2. **Negative Indexing:** It begins from the last character of a string, starting with `-1`.

Consider the below example:

```
a = "HELLO"
print(a[0])    # Result: 'H'
print(a[-1])   # Result: 'O'
```

Here, At index '0', the character 'H' is present, so Indexing starts from the left and increments to the right. At index '-1', the character 'O' is present, so Indexing starts from the right and increments to the left.

You're required to create a program that takes a **string** input from the user. Once the input is taken, perform the following actions:

1. Display the entire string.
2. Prints the second character of string from the beginning.
3. Prints the second character of string from the end.

Sample Input and Output:

Sample Test Cases

Explorer

String1.py

```
1 # Prompt the user to enter a string
2 string = str(input("String: "))
3 # Print entire string
4 print(string)
5
6 # Print second character of string from beginning
7 print("Second character from first:", string[1])
8
9 # Print second character of string from end
10 print("Second character from end:", string[-2])
```

Terminal

Test cases

30.2.2. Understanding Slicing Operator

Python provides many ways to extract a part of a string using a concept called **Slicing**. It is used to extract a specific part of a string using a **startIndex** and an **endIndex**.

The syntax for using the slice operation on a string is:

```
[startIndex : endIndex : step]
```

where,

- **startIndex** is the index where you want to start extracting.
- **endIndex** is the index where you want to stop extracting (not included).
- **step** is the interval between characters to include in the slice.

Key considerations to keep in mind when performing slicing operations:

- If you omit the **startIndex**, the slice starts from the beginning (index 0).
- If you omit the **endIndex**, the slice goes until the end of the string.
- Positive **step** means moving from left to right, incrementing the index by step.
- Negative **step** means moving from right to left, decrementing the index by step.
- **startIndex**, **endIndex** and **step** are optional parameters.

Click on the [Live Demo](#) button to know more about **Slicing in Strings**.

Let's challenge your understanding with a small question:

While using string slicing in Python, if you omit the **endIndex** parameter, the slice will include characters up to:

- ☒ The last character of the string, including it.
- ☐ The last character of the string, excluding it.
- ☐ The first character of the string.
- ☐ The character at the midpoint of the string.

30.2.3. Concatenation of strings

Concatenation of strings refers to the process of linking or combining two strings into a single string. The `(+)` operator joins the text on both sides of the operator.

Write a program to take two strings as input from the console using `input()` function. Concatenate those two strings and print the result as shown in the sample test case.

Sample Input and Output:

```
str1: Python
str2: Jython
Python Jython
```

Hint: Use `(" ")` for providing space between the two concatenated strings.

Sample Test Cases

Explorer

String3.py

```
1 str1 = str(input("str1: "))
2 str2 = str(input("str2: "))
3 print(str1 + " " + str2)
```

Terminal

Test cases

30.2.4. Program to remove a character from string based on integer value given by ... 11:43

Write a program to remove a character from a string based on integer value given by the user. Treat the input as an index of a string.

Print the result as shown in the sample test cases.

Sample Input and Output 1:

str: Python Programming

num: 9

Python Programming

Sample Input and Output 2:

str: Strings

num: 10

num should be positive, less than the length of str

Sample Test Cases

Explorer

StringTes...

```
1 # Type your content here...
2 str1 = str(input("str: "))
3 num = int(input("num: "))
4 v if num < len(str1) and num >= 0:
5     → print("output:", str1[0:num] + str1[num+1: ])
6 v else:
7     → print("num should be positive, less than the length of
8     # print("output:", result)
```

Terminal

Test cases

30.2.5. Understanding Membership Operator

01:29

Membership Operators **in** and **not in** are used to check whether a certain element exists within a sequence such as string, list, tuple or set.

- The **in** operator returns **True** if the specified element is found in the sequence. Otherwise, it returns **False**
- The **not in** operator returns **True** if the specified element is not found in the sequence. Otherwise, it returns **False**
- The results are based on case sensitivity.

Below are the examples which demonstrates the use of **in** and **not in** operators:

```
s = "good morning"
print("m" in s)           # Result: True
print("a" not in s)       # Result: True
print("z" in s)           # Result: False
print("good" in s)        # Result: True
print("Morning" in s)     # Result: False
```

Consider the following statements about the string `str = "Coding"` and select the correct options:

- ☐ `print(r in str)` returns **False**
- ☐ `print('s' in str)` returns **True**
- ☒ `print('z' not in str)` returns **True**

Write the code

Your task is to:

- Take a string from the user.
- print the distinct characters of the string separated by space.

Constraints:

1 <= length of the string <= 50

Sample Test case:

aabbccccddd --> String with the repeated characters

a b c d --> The output consists of the distinct characters

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when the user's input and output match the expected input and output.

Sample Test Cases

Test Case 1:

Expected Output:

aabbccccddd

a b c d

Test Case 2:

Expected Output:

execute

e x e c u t e

distinct.py

Submit

```
1 #write your code here.
2 str1 = str(input())
3 unique_char = set(str1)
4 print(" ".join(unique_char))
```


Write the code

Your task is to:

- Take two strings **s1** and **s2** from the user.
- Make a new string **s3** by adding both strings **s1** and **s2**
- Write a program to create a new string by appending **s1** in the middle of **s3**.

Note: Refer to the Displayed test cases for a better understanding.

Constraints:

- 1 <= length of the strings <= 50
- Middle of the string is considered as length of the string divided by 2, if middle came as float do truncation and take integer value (ex 0.5, 2.5 became 0 and 2 middle indices).

Sample Test case:

CODE ---> Input string **s1**.
 add ---> Input string **s2**.
 CODCODEEadd----> print the new string by appending **s1** in the middle of **s3**

Explanation:

Given strings s1 = CODE
 s2 = add
 s3 = CODEadd
 appending s1 in s3 results in **COD + CODE + Eadd**

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when the user's input and output match the expected input and output.

Sample Test Cases

distinct.py

```
1 #write your code here.
2 s1 = str(input())
3 s2 = str(input())
4 s3 = s1 + s2
5 length = len(s3)//2
6 print(s3[0:length] + s1 + s3[length:])
```

Execution Results

3 out of 3 shown cases successful
 4 out of 4 hidden cases successful

✓ Test Case - 1 (Execution Time: 12 ms)

Expected Output	User Output
MAin	MAin
GAin	GAin
MAinMAinGAin	MAinMAinGAin

✓ Test Case - 2 (Execution Time: 11 ms)

Expected Output	User Output
AD	AD
ABC	ABC
ADADABC	ADADABC

Finish Clear

Submit Prev Next

Write the code

Your task is to:

- Take a string from the user.
- Print the count of occurrences of all characters

Note:

- Refer to the Displayed test cases for a better understanding.
- The input string contain's any type of characters (alphabets lower or upper case, numeric characters, or special characters)

Constraints:

1 <= length of the string <= 50

Sample Test case:

app
a : 1
p : 2

Explanation:

In the given test case,
The occurrence of **a** = 1
The occurrence of **p** = 2
So the output prints as required.

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when the user's input and output match the expected input and output.

Sample Test Cases

distinct.py

```
1 #write your code here.  
2 string = input()  
3 occurence = {}  
4 for char in string:  
5     if char in occurence:  
6         occurence[char] += 1  
7     else:  
8         occurence[char] = 1  
9  
10 for char , count in occurence.items():  
11     print(char , ":", count)  
12  
13
```

Execution Results

5 out of 5 shown cases successful

4 out of 4 hidden cases successful

✓ Test Case - 1 (Execution Time: 6 ms)

Expected Output

app
a : 1
p : 2

User Output

app
a : 1
p : 2

✓ Test Case - 2 (Execution Time: 8 ms)

Expected Output

execute
e : 3
x : 1
c : 1
u : 1

User Output

execute
e : 3
x : 1
c : 1
u : 1

Finish

Clear

Submit

Prev

Next

Write the code

Your task is to:

- Take a string from the user.
- Create a string made of the middle and last characters of the given string.

Note: Use `x = len(str) / 2` to get the middle character.

Constraints:

1 <= length of the string <= 50

Sample Test case:

python--> Input String.

hn --> Print the string made of the middle and last character

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when the user's input and output match the expected input and output.

Sample Test Cases

Test Case 1:

Expected Output:

python

hn

Test Case 2:

Expected Output:

pending

distinct.py

Submit

```
1 #write your code here.
2 str1 = str(input())
3 x = len(str1)/2
4
5 y = str1[-1:0]
6 result = y
7 print(result)
8
9
```

Execution Results

0 out of 2 shown cases successful

0 out of 5 hidden cases successful

Show only failed cases

✖ Test Case - 1 (Execution Time: 7 ms)


Expected Output

python

hn

User Output

python

⚠  : indicates the mismatch in the expected output.

✖ Test Case - 2 (Execution Time: 7 ms)


Expected Output

pending

dg

User Output

pending

⚠  : indicates the mismatch in the expected output.

Finish

Clear

Submit

Prev

Next

Write the code

Your task is to:

- Take a string from the user.
- print the distinct characters of the string separated by space.

Constraints:

1 <= length of the string <= 50

Sample Test case:

aabbccccddd ----> String with the repeated characters

a b c d ----> The output consists of the distinct characters

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when the user's input and output match the expected input and output.

Sample Test Cases

Test Case 1:

Expected Output:

aabbccccddd

a b c d

Test Case 2:

Expected Output:

execute

e x e c u t e

distinct.py

```
1 #write your code here.
2 str1 = str(input())
3 unique_char = set(str1)
4 print(" ".join(unique_char))
```

Submit

Execution Results

0 out of 2 shown cases successful

0 out of 4 hidden cases successful

Show only failed case

✖ Test Case - 1 (Execution Time: 8 ms)

Expected Output

aabbccccddd

a b c d

User Output

aabbccccddd

d b c a

⚠ indicates the mismatch in the expected output.

✖ Test Case - 2 (Execution Time: 6 ms)

Expected Output

execute

e x e c u t e

User Output

execute

x t a e c

⚠ indicates the mismatch in the expected output.

Finish

Clear

Submit

Prev

Next

Write the code

Your task is to:

- Take s string from the user.
- Given string contains a combination of the lower and upper case letters.
- Write a program to arrange the characters of a string so that all uppercase letters should come first.

Note: Refer the Displayed test cases for the better understanding.

Constraints:

- $1 \leq \text{length of the string} \leq 50$
- Order of occurrence of characters will not disturb while rearranging the characters.

Sample Test case:

PyThoN ----> Input string .

PTNyho ----> Print the string after arranging the character of a string in such a way that all upper case letters should come first.

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when the user's input and output match the expected input and output.

Sample Test Cases

Test Case 1:

Expected Output:

PyThoN

PTNyho

distinct.py

```
1 #write your code here.
2 str1 = str(input())
3 uperr = ""
4 lowerr = ""
5
6 for char in str1:
7     if char.islower():
8         lowerr+=char
9     else:
10        uperr+=char
11 print(uperr+lowerr)
```

Execution Results

4 out of 4 shown cases successful

5 out of 5 hidden cases successful

✓ Test Case - 1 (Execution Time: 9 ms)

Expected Output

PyThoN

PTNyho

User Output

PyThoN

PTNyho

✓ Test Case - 2 (Execution Time: 7 ms)

Expected Output

coDEtaNtrA

DENAcotatr

User Output

coDEtaNtrA

DENAcotatr

✓ Test Case - 3 (Execution Time: 7 ms)

Submit Prev Next