STRINGS

DR. VEENA R S
ASSOCIATE PROFESSOR
DEPARTMENT OF ISE
DSATM, BENGALURU

Escape Characters

Escape character	Prints as
`\'`	Single quote
`\"`	Double quote
`\t`	Tab
`\n`	Newline (line break)
`\\`	Backslash

Raw Strings

- You can place an r before the beginning quotation mark of a string to make it a raw string.
- A raw string completely ignores all escape characters and prints any backslash that appears in the string.

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

- Print('that is carol\'s cat')

  Output:

  that is carol's cat

Multiline Strings with Triple Quotes

- A multiline string in Python begins and ends with either three single quotes or three double quotes.
- Any quotes, tabs, or newlines in between the “triple quotes” are considered part of the string.

Multiline Comments

- While the hash character (#) marks the beginning of a comment for the rest of the line.
- A multiline string is often used for comments that span multiple lines.

```
"""This is a test Python program.
Written by Al Sweigart al@inventwithpython.com

This program was designed for Python 3, not Python 2.
"""
```

Indexing and Slicing Strings

- Strings use indexes and slices the same way lists do.

'Hello world!'

H	e	l	l	o		w	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

The ‘in’ and ‘not in’ Operators with Strings

- The in and not in operators can be used with strings just like with list values.

```
>>> 'Hello' in 'Hello World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

PUTTING STRINGS INSIDE OTHER STRINGS

1. Using + (Concatenation Operator)

Eg:

```
>>> name = 'Al'
>>> age = 4000
>>> 'Hello, my name is ' + name + '. I am ' + str(age) + ' years old.'
```

'Hello, my name is Al. I am 4000 years old.‘

This requires a lot of tedious typing

PUTTING STRINGS INSIDE OTHER STRINGS

2. Using **String interpolation,** in which the %s operator inside the string acts as a marker to be replaced by values following the string.

One benefit of string interpolation is that str() doesn't have to be called to convert values to strings.

Eg:

```
>>> name = 'Al'
>>> age = 4000
>>> 'My name is %s. I am %s years old.' % (name, age)
'My name is Al. I am 4000 years old.'
```

PUTTING STRINGS INSIDE OTHER STRINGS

2. Using **f-strings,** which is similar to string interpolation except that braces are used instead of %s, with the expressions placed directly inside the braces.

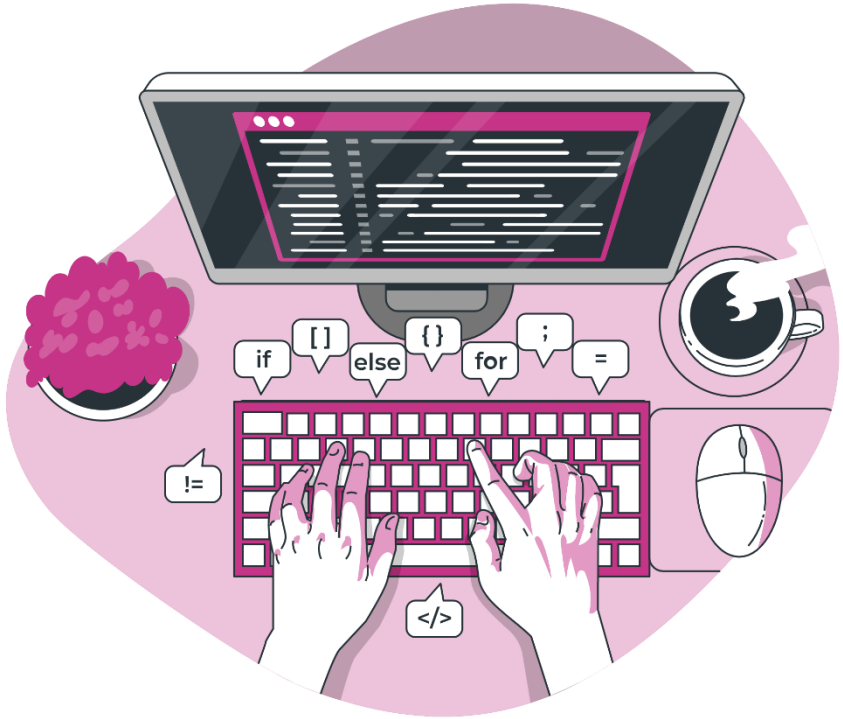
f-strings have an f prefix before the starting quotation mark.

Eg:

```
>>> name = 'Al'
>>> age = 4000
>>> f'My name is {name}. Next year I will be {age + 1}.'
'My name is Al. Next year I will be 4001.'
```

Useful String Methods

- upper()
- lower()
- isupper()
- islower()
- Xstring
- isalpha()
- isalnum()
- isdecimal()
- isspace()
- istitle()
- startswith()
- endswith()
- join()
- split()
- rjust()
- ljust()
- center()
- strip()
- rstrip()
- lstrip()



3. Aim: Demonstration of manipulation of strings using string methods.

a) Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters.

```
s = input("Enter a sentence: ")
w, d, u, l = 0, 0, 0, 0
l_w = s.split()
w = len(l_w)
for c in s:
        if c.isdigit():
                d = d + 1
        elif c.isupper():
                u = u + 1
        elif c.islower():
                l = l + 1
print ("No of Words: ", w)
print ("No of Digits: ", d)
print ("No of Uppercase letters: ", u)
print ("No of Lowercase letters: ", l)
```

b) Write a Python program to find the string similarity between two given strings. Sample Output: Sample Output:

Original string: Original string:

Python Exercises Python Exercises

Python Exercises Python Exercise

Similarity between two said strings: Similarity between two said strings:

1.0 0.967741935483871

```
import difflib
def string_similarity(str1, str2):
        result = difflib.SequenceMatcher(a=str1.lower(), b=str2.lower())
        return result.ratio()
str1 = input("Enter the first string")
str2 = input("Enter the second string")
print("Original strings:")
print(str1)
print(str2)
print("Similarity between two said strings:")
print(string_similarity(str1,str2))
```