Name :- Deepawali . B. Mhaisagar
Assignment no 6

1.What are escape characters, and how do you use them?
ANS
Escape characters are special characters in programming languages, including Python, that are used to represent certain non-printable or special characters within strings. They are denoted by a backslash (\) followed by a specific character or sequence.

Here are some commonly used escape characters in Python:

- \n: Newline character. It inserts a newline in the string, starting a new line.
- \t: Tab character. It inserts a horizontal tab in the string.
- \\: Backslash character. It represents a literal backslash in the string.
- \': Single quote character. It represents a literal single quote within a string enclosed in single quotes.
- \": Double quote character. It represents a literal double quote within a string enclosed in double quotes.

```
message = 'Hello\nWorld'

print(message)
```

O/P

Hello

World

2. What do the escape characters n and t stand for?

ANS

In Python, the escape character \n represents the newline character, and the escape character \t represents the tab character.

Here's a breakdown of what these escape characters stand for:

- \n: Newline character
  - The escape character \n represents a newline. It is used to insert a line break in a string, starting a new line.
  - When encountered in a string, the \n escape sequence is replaced with the actual newline character, causing the text to move to the next line.
  - For example:

```
print("Hello\nWorld")
```

O/P

Hello

World

\t: Tab character

- The escape character \t represents a horizontal tab.
- It is used to insert a tabulation, which typically represents multiple spaces, in a string.
- When encountered in a string, the \t escape sequence is replaced with the tab character, creating horizontal spacing.
- For example:

print("Name:\tJohn")

O/P

**Name:	John**

3.What is the way to include backslash characters in a string?

ANS

To include a backslash character (\) itself in a string, you can use a double backslash (\\). The double backslash acts as an escape sequence, with the first backslash escaping the second one, resulting in a single backslash character being included in the string.

Here's an example to illustrate how to include a backslash in a string:

message = "This is a double backslash: \\"

print(message)

O/P

This is a double backslash: \

Alternatively, you can also define the string as a raw string by prefixing it with the r character. In a raw string, backslashes are treated as literal characters and are not interpreted as escape sequences.

message = r"This is a raw string with a backslash: \"

print(message)

O/P

```
This is a raw string with a backslash: \"
```

4. The string “Howl’s Moving Castle” is a correct value. Why isn’t the single quote character in the word How’s not escaped a problem?

ANS
In Python, the single quote character (') can be included in a string without being escaped as long as the string itself is enclosed in double quotes ("), or vice versa. This allows for more flexibility when dealing with strings that contain quotes.

In the example you provided, the string "Howl's Moving Castle" is a correct value without needing to escape the single quote character. This is because the string is enclosed in double quotes, and the single quote within the string does not conflict with the enclosing quotes.

Python allows for both single quotes and double quotes to delimit strings, and you can freely use one type of quote inside the other without needing to escape them. This flexibility makes it easier to include quotes within strings and improves readability.

Here are a few examples to illustrate this concept

string1 = "He said, 'Hello!'"

string2 = 'She replied, "Hi!"'

string3 = "I can include the single quote character ('), and it's not a problem."

string4 = 'I can include the double quote character (") without any issues.'

In the above examples, both single quotes and double quotes are used to enclose the strings, and there is no need to escape the quotes within the string because they do not conflict with the enclosing quotes.

However, if you need to include the same type of quote character within a string that is enclosed by the same type of quotes, you would need to escape it using a backslash. For example:

string = 'He said, \'Hello!\"

5. How do you write a string of newlines if you don't want to use the n character?

ANS

If you don't want to use the newline character (\n) to create a string with newlines, you can use multi-line string literals or string concatenation to achieve the desired result.

Multi-line string literals:

You can create a multi-line string by enclosing the text within triple quotes (''' or """). Each line break within the triple quotes will be treated as a newline character in the resulting string.

```
message = '''Line 1

Line 2

Line 3'''

print(message)
```

O/P

Line 1

Line 2

Line 3

6. What are the values of the given expressions?

‘Hello, world!’[1]

‘Hello, world!’[0:5]

‘Hello, world!’[:5]

‘Hello, world!’[3:]

ANS

So, the values of the given expressions are:

```
'e'
'Hello'
'Hello'
'lo, world!'
```

7. What are the values of the following expressions?

‘Hello’.upper() ANS HELLO

‘Hello’.upper().isupper() True

‘Hello’.upper().lower() False

8. What are the values of the following expressions?

‘Remember, remember, the fifth of July.’.split()

‘-’.join(‘There can only one.’.split())

ANS

Let's evaluate the given expressions:

'Remember, remember, the fifth of July.'.split() splits the string at each whitespace character and returns a list of the resulting substrings. The resulting list would be ['Remember,', 'remember,', 'the', 'fifth', 'of', 'July.'].
'There can only one.'.split() splits the string at each whitespace character and returns a list of the resulting substrings. The resulting list would be ['There', 'can', 'only', 'one.'].
'-'.join(['There', 'can', 'only', 'one.']) joins the elements of the list using the hyphen ('-') as a separator. The resulting string would be 'There-can-only-one.'.

Therefore, the values of the given expressions are:

['Remember,', 'remember,', 'the', 'fifth', 'of', 'July.']
'There-can-only-one.'

9. What are the methods for right-justifying, left-justifying, and centering a string?

ANS

In Python, strings provide several methods for adjusting the alignment of text. Here are the methods for right-justifying, left-justifying, and centering a string:

Right-justifying a string:

- str.rjust(width, fillchar): This method returns a right-justified version of the string by padding it on the left with the specified fillchar (default is space) to make it width characters long.

Left-justifying a string:

- str.ljust(width, fillchar): This method returns a left-justified version of the string by padding it on the right with the specified fillchar (default is space) to make it width characters long.

Centering a string:

- str.center(width, fillchar): This method returns a centered version of the string by padding it equally on both sides with the specified fillchar (default is space) to make it width characters long.

Here's an example demonstrating the usage of these methods:

```
text = "Hello"

width = 10

fillchar = "*"



right_justified = text.rjust(width, fillchar)

left_justified = text.ljust(width, fillchar)

centered = text.center(width, fillchar)



print("Right-justified:", right_justified)

print("Left-justified:", left_justified)

print("Centered:", centered)
```

O/P

```
Right-justified: *****Hello

Left-justified: Hello*****

Centered: **Hello***
```

In the example above, the width parameter determines the total width of the resulting string, and the fillchar parameter specifies the character used for padding.

10. What is the best way to remove whitespace characters from the start or end?

ANS

To remove whitespace characters from the start or end of a string in Python, you can use the following methods:

- str.strip(): This method returns a new string with leading and trailing whitespace characters removed.
- str.lstrip(): This method returns a new string with leading (left) whitespace characters removed.
- str.rstrip(): This method returns a new string with trailing (right) whitespace characters removed.

Here's an example to illustrate the usage of these methods:

```
text = "   Hello, world!   "



# Removing leading and trailing whitespace

trimmed = text.strip()



# Removing leading whitespace

left_trimmed = text.lstrip()



# Removing trailing whitespace

right_trimmed = text.rstrip()



print("Original string:", text)

print("Trimmed string:", trimmed)

print("Left-trimmed string:", left_trimmed)

print("Right-trimmed string:", right_trimmed)
```

As shown in the example, str.strip() removes leading and trailing whitespace, str.lstrip() removes leading whitespace, and str.rstrip() removes trailing whitespace. These methods return new strings, leaving the original string unchanged.