ASSIGNMENT - 6 (PYTHON)

1. What are escape characters, and how do you use them?

Escape characters are special characters that are used to represent non-printable or reserved characters within strings in programming languages or markup languages. They allow you to include characters in strings that would otherwise be difficult or impossible to type directly.

In most programming languages, escape characters are preceded by a backslash (). When the compiler or interpreter encounters an escape character in a string, it treats the character following the backslash in a special way.

Here are some common escape characters and their meanings:

1. **\n:** Newline character. It moves the cursor to the beginning of the next line.
2. **\t:** Tab character. It inserts a tab space.
3. **\':** Single quote. It allows you to include a single quote within a string that is enclosed in single quotes.
4. **\":** Double quote. It allows you to include a double quote within a string that is enclosed in double quotes.
5. **\\:** Backslash. It allows you to include a backslash character in a string.

Example:

```
print("This is a string with a newline character: \nThis is the second line.")

print("This is a string with a tab character: \tTabbed text.")

print("This is a string with a single quote: \'Single quote inside single quotes.")

print("This is a string with a double quote: \"Double quote inside double quotes.")

print("This is a string with a backslash: \\")
```

Output:

This is a string with a newline character:

This is the second line.

This is a string with a tab character:	Tabbed text.

This is a string with a single quote: 'Single quote inside single quotes.

This is a string with a double quote: "Double quote inside double quotes.

This is a string with a backslash: \

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

The escape characters \n and \t stand for newline and tab, respectively.

\n: Newline character. When encountered within a string, it moves the cursor to the beginning of the next line.

\t: Tab character. It inserts a tab space, which typically represents a horizontal tabulation or indentation in text.

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

To include a backslash character (\) in a string, you need to use a double backslash (\\). This is because the backslash is an escape character itself, so to represent a literal backslash, we need to escape it by using another backslash.

Example:

print("This is a backslash: \\")

Output:

This is a backslash: \

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

In programming languages like Python, strings can be enclosed in either single quotes (') or double quotes ("). This flexibility allows you to include single quotes or double quotes within strings without needing to escape them if the string is enclosed in the opposite type of quote.

So in the case of the string "Howl's Moving Castle", the string is enclosed in double quotes, which means that the single quote within the word "Howl's" doesn't need to be escaped. The Python interpreter understands that the single quote is part of the string and not the delimiter for the end of the string.

Similarly, if you were to enclose the string in single quotes instead, like this: 'Howl\'s Moving Castle', then the single quote within "Howl's" would need to be escaped with a backslash (\') because the string is now delimited by single quotes, and including an unescaped single quote would prematurely terminate the string.

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

If you want to write a string with newlines without using the `\n` escape sequence, you can use triple quotes (`'''` or `"""`) in Python. Triple quotes allow you to create multiline strings where newlines are included directly in the string without the need for escape characters.

Example:

```
multiline_string = '''This is line 1.

This is line 2.

This is line 3.'''

print(multiline_string)
```

Output:

```
This is line 1.

This is line 2.

This is line 3.
```

You can also use double quotes for triple-quoted strings:

```
multiline_string = """This is line 1.

This is line 2.

This is line 3."""

print(multiline_string)
```

Output:

This is line 1.

This is line 2.

This is line 3.

6. What are the values of the given expressions?

'Hello, world !'[1]

'Hello, world !'[0:5]

'Hello, world !'[:5]

'Hello, world !'[3:]

The values of the given expressions are:

1. **'e'**
2. **'Hello'**
3. **'Hello'**
4. **'lo, world !'**

7. What are the values of the following expressions?

'Hello'.upper()

'Hello'.upper().isupper()

'Hello'.upper().lower()

The values of the given expressions are:

“HELLO”

‘True’

“hello”

8. What are the values of the following expressions?

'Remember, remember, the fifth of July .'.split()

'-'.join('There can only one'.split())

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?

1. 'str.ljust(width[, fillchar])': This method left-justifies a string in a field of given width. If the string is shorter than the width, it is padded with spaces (or optionally with a specified fill character) on the right.

Example:

```
text = "Hello"

justified_text = text.rjust(10)

print(justified_text)  # Output: "     Hello"
```

2. 'str.ljust(width[, fillchar])': This method left-justifies a string in a field of given width. If the string is shorter than the width, it is padded with spaces (or optionally with a specified fill character) on the right.

Example:

```
text = "Hello"

justified_text = text.ljust(10)

print(justified_text)  # Output: "Hello     "
```

3. 'str.center(width[, fillchar])': This method centers a string in a field of given width. If the string is shorter than the width, it is padded with spaces (or optionally with a specified fill character) on both sides.

Example:

```
text = "Hello"

justified_text = text.center(10)
```

```
print(justified_text)  # Output: "  Hello   "
```

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

The best way to remove whitespace characters from the start or end of a string in Python is to use the `str.strip()` method.

```
text = "   Hello, world!   "

stripped_text = text.strip()

print(stripped_text)  # Output: "Hello, world!"
```

The `strip()` method removes leading and trailing whitespace characters (spaces, tabs, newlines, etc.) from the string. If you only want to remove whitespace characters from the start or end, you can use `str.lstrip()` to remove leading whitespace or `str.rstrip()` to remove trailing whitespace.