**Python Assignment 6**

Q1. What are escape characters, and how do you use them?

Ans. In Python, escape characters are special characters that are used to represent certain characters or symbols that are difficult to type or require special formatting. They are used within string literals and are indicated by a backslash ("") followed by a character or sequence of characters. Here are some common escape characters and their meanings in Python:

\n - Newline character

\t - Tab character

\b - Backspace character

\r - Carriage return character

' - Single quote character

" - Double quote character

To use an escape character in Python, simply include the backslash followed by the appropriate character or sequence of characters within a string or character literal. For example, to include a newline character in a string literal, you would write:

print("Hello\nWorld!")

This would output:

Hello

World!

Similarly, to include a double quote within a string literal, you would use the escape character like this:

print("She said, \"Hello!\"")

This would output:

She said, "Hello!"

Escape characters are a useful tool for formatting and manipulating strings in Python.

Q2. What do the escape characters n and t stand for?

Ans. In programming, the escape character "\n" represents a newline character and "\t" represents a tab character.

The "\n" character is used to insert a line break or start a new line in a string. For example, if you wanted to print the following text on separate lines:

Hello

World!

You would use the escape character "\n" like this:

print("Hello\nWorld!")

The "\t" character is used to insert a horizontal tab or indentation in a string. For example, if you wanted to print the following text with a tab between "Hello" and "World":

Hello World!

You would use the escape character "\t" like this:

print("Hello\tWorld!")

Escape characters are a useful tool for formatting strings and controlling how they are displayed in output.

Q3. What is the way to include backslash characters in a string?

Ans. To include a backslash character "" in a string in Python, you need to use the escape character "\".

The backslash is used as an escape character in Python, so if you simply try to include a backslash in a string without escaping it, it will be interpreted as the start of an escape sequence.For example, if you wanted to include the following string in Python:

C:\Users\John

You would need to use the escape character "\" before each backslash in the string like this:

print("C:\\Users\\John")

This would output:

C:\Users\John

Similarly, in other programming languages, such as JavaScript or Java, you would use the same approach to include a backslash character in a string.

Using the escape character "\" is the standard way to include a backslash character in a string in Python.

Q4. 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?

Ans. The string "Howl's Moving Castle" is a correct value because the single quote character in the word "Howl's" is not causing a problem due to the use of double quotes to define the string. In Python (and many other programming languages), strings can be defined using single quotes ('...') or double quotes ("..."). If you use double quotes to define the string, you can include single quotes within the string without escaping them, and vice versa.

For example, the following two string definitions are equivalent:

title1 = "Howl's Moving Castle"

title2 = 'Howl\'s Moving Castle'

Both of these strings will be interpreted as "Howl's Moving Castle". In the first example, the single quote within the string is not causing a problem because the string is defined using double quotes. In the second example, the single quote within the string is escaped using a backslash because the string is defined using single quotes.

So, in short, the single quote character in "Howl's" is not causing a problem in this case because it is not being interpreted as a delimiter for the string due to the use of double quotes to define the string.

Q5. 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 "\n" character to represent newlines in a string in Python, you can use multiline strings and separate the lines using triple quotes ("""...""").

For example, if you wanted to create a string with three lines of text:

This is line 1.

This is line 2.

This is line 3.

You could define it as a multiline string like this:

string\_without\_n = """This is line 1.

This is line 2.

This is line 3."""

This would create a string with three lines of text, without using the "\n" character. Note that you can also use a backslash "" to escape the end-of-line character and continue the string on the next line, like this:

string\_without\_n = "This is line 1.\

This is line 2.\

This is line 3."

However, this can make the code less readable and harder to maintain, so using a multiline string is generally a better option.

Q6. What are the values of the given expressions?

'Hello, world!'[1]

'Hello, world!'[0:5]

'Hello, world!'[:5]

'Hello, world!'[3:]

Ans. The values of the given expressions are:

'Hello, world!'[1] returns the character at index 1, which is 'e'.

'Hello, world!'[0:5] returns the substring from index 0 up to (but not including) index 5, which is 'Hello'.

'Hello, world!'[:5] is the same as the previous expression, and also returns 'Hello'.

'Hello, world!'[3:] returns the substring from index 3 to the end of the string, which is 'lo, world!'.

Q7. What are the values of the following expressions?

'Hello'.upper()

'Hello'.upper().isupper()

'Hello'.upper().lower()

Ans. The values of the following expressions are:

'Hello'.upper() returns the string in uppercase, which is 'HELLO'.

'Hello'.upper().isupper() returns True because the string 'HELLO' is all uppercase.

'Hello'.upper().lower() returns the string in lowercase, which is 'hello'.

So, the overall values of the expressions are:

'Hello'.upper() returns 'HELLO'.

'Hello'.upper().isupper() returns True.

'Hello'.upper().lower() returns 'hello'.

Q8. What are the values of the following expressions?

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

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

Ans. The values of the following expressions are:

'Remember, remember, the fifth of July.'.split() splits the string at whitespace characters (spaces, tabs, newlines, etc.) and returns a list of the resulting substrings, which is ['Remember,', 'remember,', 'the', 'fifth', 'of', 'July.'].

'-'.join('There can only one.'.split()) splits the string at whitespace characters and then joins the resulting substrings using the dash character -. The resulting string is 'There-can-only-one.'.

So, the overall values of the expressions are:

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

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

Q9. What are the methods for right-justifying, left-justifying, and centering a string?

Ans. In Python, the following are the methods for right-justifying, left-justifying, and centering a string: str.rjust(width, [fillchar]): Returns a string with the original string right-justified to a total width. You can optionally specify a fill character to use in place of whitespace. For example:

>>> s = "hello"

>>> s.rjust(10, "-")

'-----hello'

str.ljust(width, [fillchar]): Returns a string with the original string left-justified to a total width. You can optionally specify a fill character to use in place of whitespace. For example:

>>> s = "hello"

>>> s.ljust(10, "-")

'hello-----'

str.center(width, [fillchar]): Returns a string with the original string centered in a total width. You can optionally specify a fill character to use in place of whitespace. For example:

>>> s = "hello"

>>> s.center(10, "-")

'--hello---'

In all of these methods, width is the total width of the resulting string, and fillchar is the optional fill character to use. If you do not specify a fill character, the method will use whitespace as the default fill character.

Q10. What is the best way to remove whitespace characters from the start or end?

Ans. In Python, you can use the strip() method to remove whitespace characters from the start and end of a string. The strip() method removes any whitespace characters such as spaces, tabs, and newlines from the beginning and end of the string. Here's an example:

>>> s = " Hello, world! "

>>> s.strip()

'Hello, world!'

In this example, the strip() method removes the two whitespace characters at the start and end of the string, leaving only the desired text.

If you only want to remove whitespace characters from the start or end of the string, you can use lstrip() or rstrip() respectively. lstrip() removes whitespace characters from the start of the string, and rstrip() removes whitespace characters from the end of the string. Here are some examples:

>>> s = " Hello, world! "

>>> s.lstrip()

'Hello, world! '

>>> s.rstrip()

' Hello, world!'

In these examples, lstrip() removes the whitespace characters at the start of the string, and rstrip() removes the whitespace characters at the end of the string.