**FSDS MAY BATCH 2022(Python Basics 6)**

**Submitted by: Shubham Tiwari**

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

Ans: Escape characters are special characters in programming that are used to represent characters that can't be entered directly into a string literal. They are represented by a backslash (**\**) followed by another character. For example, the escape character ‘**\n’** represents a newline, ‘**\t’** represents a tab, and ‘**\\’** represents a backslash.

Escape characters are used in string literals to represent characters that are difficult or impossible to enter directly. They are widely used in programming languages, including Python, Java, C, and others. To use an escape character, simply place a backslash (**\**) before the character you want to escape.

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

Ans: The escape character ‘**\n’** represents a newline, ‘**\t’** represents a tab.

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

Ans: To include a backslash (‘**\’**) character in a string, you need to escape it by using two consecutive backslashes (‘**\\’**). This tells the programming language that we want to include a literal backslash character in the string, rather than using it as an escape character.

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 it uses double quotes (**"**) to define the string, and the single quote character (**'**) inside the string is not being used as a string delimiter.

In Python, the following string is defined using double quotes, and the single quote character inside the string does not need to be escaped:

**string = "Howl's Moving Castle"**

Q5. How do you write a string of newlines if you don’t want to use the n character?

Ans: If we don't want to use the ‘**\n’** escape character to represent newlines in a string, you can use a different method that is specific to the programming language you are using.

For example, in Python, we can use triple quotes (**"""** or **'''**) to define a multi-line string, and newline characters in the string will be preserved:

**string = """**

**This is the first line.**

**This is the second line.**

**"""**

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 in Python are:

1. **'Hello, world!'[1]** returns the second character of the string, which is **'e'**.
2. **'Hello, world!'[0:5]** returns a slice of the string, which includes the characters from the first to the fifth (not including the fifth), i.e., **'Hello'**.
3. **'Hello, world!'[:5]** is equivalent to **'Hello, world!'[0:5]** and returns the same result.
4. **'Hello, world!'[3:]** returns a slice of the string, which includes all the characters from the fourth to the end of the string, i.e., **'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 in Python are:

1. **'Hello'.upper()** returns the uppercase version of the string **'Hello'**, which is **'HELLO'**.
2. **'Hello'.upper().isupper()** returns a Boolean value indicating whether the result of **'Hello'.upper()** is an uppercase string. Since **'HELLO'** is an uppercase string, this expression returns **True**.
3. **'Hello'.upper().lower()** returns the lowercase version of the result of **'Hello'.upper()**, which is **'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 in Python are:

1. **'Remember, remember, the fifth of July'.split()** returns a list of words obtained by splitting the original string on whitespaces, i.e., **['Remember,', 'remember,', 'the', 'fifth', 'of', 'July']**.
2. **' - '.join('There can only one '.split())** returns a string obtained by joining the list of words obtained by splitting the string **'There can only one '** on whitespaces using the separator **' - '**, i.e., **'There - can - only - one'**.

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

Ans: In Python, we can right-justify, left-justify, and center a string using the **str.rjust()**, **str.ljust()**, and **str.center()** methods, respectively.

1. **str.rjust(width)** returns a right-justified string of length **width**, padding the original string with spaces on the left as necessary.
2. **str.ljust(width)** returns a left-justified string of length **width**, padding the original string with spaces on the right as necessary.
3. **str.center(width)** returns a centered string of length **width**, padding the original string with spaces on both sides as necessary.

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

Ans: In Python, the easiest and most efficient way to remove whitespace characters from the start or end of a string is to use the **str.strip()** method.

**str.strip()** returns a copy of the string with leading and trailing whitespace characters removed.

For example:

**string = " Hello, world! "**

**print(string.strip())**

**str.strip()** removes only leading and trailing whitespaces. If we need to remove whitespaces from the middle of the string, you can use the str.replace() method or split and join the string.