1. What are escape characters, and how do you use them?In programming, an escape character is a special character that is used to represent other characters that cannot be directly typed or included in a string. In Python, escape characters are denoted by the backslash () symbol followed by another character or sequence of characters. Here are some commonly used escape characters in Python: \n: Newline character. It is used to insert a new line in the string. \t: Tab character. It is used to insert a tab in the string. ': Single quote character. It is used to insert a single quote in the string. ": Double quote character. It is used to insert a double quote in the string.

In [1]:
```
#example of using escape characters in Python:
print("This is a string with a\nnewline character.")
print("This is a string with a\ttab character.")
print("This is a string with a\'single quote character.")
print("This is a string with a\"double quote character.")
print("This is a string with a\\backslash character.")
```

```
This is a string with a
newline character.
This is a string with a tab character.
This is a string with a'single quote character.
This is a string with a"double quote character.
This is a string with a\backslash character.
```

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

In Python, the escape character \n stands for a newline character, and the escape character \t stands for a tab character.

The \n character is used to create a new line in a string. When we use this escape character in a string, it tells Python to insert a line break at that point in the string, which creates a new line

The \t character is used to insert a tab in a string. When we use this escape character in a string, it tells Python to insert a horizontal tab at that point in the string.

In [2]:
```
#example of \n
print("This is the first line.\nThis is the second line.")
#example of \t
print("First column\tSecond column\tThird column")
```

```
This is the first line.
This is the second line.
First column    Second column   Third column
```

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

To include a backslash character () in a string in Python, we can use the backslash character as an escape character.

This means that we sould use two backslashes (\) to represent a single backslash character in the string

In [3]:
```
print("This is a backslash: \\")
```

```
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 Python, we can enclose a string in single quotes or double quotes. When a string is enclosed in single quotes, we can include double quotes within the string without any problem. Similarly,

when a string is enclosed in double quotes, we can include single quotes within the string without any problem.

In the given string "Howl's Moving Castle", the string is enclosed in double quotes, and the word "Howl's" contains a single quote character. Since the string is enclosed in double quotes, the single quote character within the string is not interpreted as the end of the string. Instead, the single quote character is treated as a regular character within the string.

Therefore, there is no need to escape the single quote character in the word "Howl's" in the given string. However, if the string was enclosed in single quotes and it contained a single quote character, then you would need to escape the single quote character using the backslash character (\') to avoid syntax errors.

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

If we don't want to use the \n escape character to write a string of newlines in Python, we can use a multiline string literal by enclosing the string in triple quotes (""" or ''').

Within a multiline string, we can include newlines simply by pressing Enter to start a new line.

In [4]:
```
multiline_string = """This is a string
that contains
multiple lines."""
print(multiline_string)
```

```
This is a string
that contains
multiple lines.
```

6. What are the values of the given expressions? 'Hello, world!'[1] 'Hello, world!'[0:5] 'Hello, world!'[:5] 'Hello, world!'[3:]

In [12]:
```
print('Hello, world!'[1])
print('Hello, world!'[0:5])
print('Hello, world!'[:5])
print('Hello, world!'[3:])
```

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

7. What are the values of the following expressions? 'Hello'.upper() 'Hello'.upper().isupper() 'Hello'.upper().lower()

In [9]:
```
'Hello'.upper()
```

Out[9]:
```
'HELLO'
```

In [10]:
```
'Hello'.upper().isupper()
```

Out[10]:
```
True
```

In [11]:
```
'Hello'.upper().lower()
```

Out[11]:
```
'hello'
```

8. What are the values of the following expressions? 'Remember, remember, the fifth of July.'.split() '-'.join('There can only one.'.split())

In [13]:
```
'Remember, remember, the fifth of July.'.split()
```

Out[13]:
```
['Remember,', 'remember,', 'the', 'fifth', 'of', 'July.']
```

In [14]:
```
'-'.join('There can only one.'.split())
```

Out[14]:
```
'There-can-only-one.'
```

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

In Python, you we use the following methods to justify a string:

str.ljust(width[, fillchar]): This method returns the original string left-justified in a string of specified width. If the original string is shorter than the specified width, it is padded with the optional fill character (default is space) on the right side.

str.rjust(width[, fillchar]): This method returns the original string right-justified in a string of specified width. If the original string is shorter than the specified width, it is padded with the optional fill character (default is space) on the left side.

str.center(width[, fillchar]): This method returns the original string centered in a string of specified width. If the original string is shorter than the specified width, it is padded with the optional fill character (default is space) on both sides.

In [15]:
```
string = "Hello"

# left-justify string in a field of width 10
left_justified = string.ljust(10)
print(left_justified)

# right-justify string in a field of width 10
right_justified = string.rjust(10)
print(right_justified)

# center string in a field of width 10
centered = string.center(10)
print(centered)
```

```
Hello
     Hello
  Hello
```

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

In Python, the best way to remove whitespace characters (spaces, tabs, newlines) from the start or end of a string is to use the strip() method.

The strip() method removes any whitespace characters from the beginning and end of the string and returns the modified string.

In [17]:
```
string = "  Hello World!   \t\n"
stripped_string = string.strip()
print(stripped_string)
```

```
Hello World!
```

In [ ]: