Python Raw Strings

Summary: in this tutorial, you will learn about the Python raw strings and how to use them to handle strings that treat the backslashes as literal characters.

Introduction the Python raw strings

In Python, when you prefix a string with the letter `r` or `R` such as `r'...'` and `R'...'`, that string becomes a raw string. Unlike a regular string, a raw string treats the backslashes (`\`) as literal characters.

Raw strings are useful when you deal with strings that have many backslashes, for example, regular expressions (https://www.pythontutorial.net/python-regex/python-regular-expressions/) or directory paths on Windows.

To represent special characters such as tabs and newlines, Python uses the backslash (`\`) to signify the start of an escape sequence. For example:

```
s = 'lang\tver\nPython\t3'
print(s)
```

Output:

```
lang    ver
Python  3
```

However, raw strings treat the backslash (`\`) as a literal character. For example:

```
s = r'lang\tver\nPython\t3'
print(s)
```

Output:

```
lang\tver\nPython\t3
```

A raw string is like its regular string with the backslash (`\`) represented as double backslashes (`\\`):

```
s1 = r'lang\tver\nPython\t3'
s2 = 'lang\\tver\\nPython\\t3'

print(s1 == s2) # True
```

In a regular string, Python counts an escape sequence as a single character:

```
s = '\n'
print(len(s)) # 1
```

However, in a raw string, Python counts the backslash (`\`) as one character:

```
s = r'\n'
print(len(s)) # 2
```

Since the backslash (`\`) escapes the single quote (`'`) or double quotes (`"`), a raw string cannot end with an odd number of backslashes.

For example:

```
s = r'\'
```

Error:

```
SyntaxError: EOL while scanning string literal
```

Or

```
s = r'\\\'
```

Error:

```
SyntaxError: EOL while scanning string literal
```

Use raw strings to handle file path on Windows

Windows OS uses backslashes to separate paths. For example:

```
c:\user\tasks\new
```

If you use this path as a regular string, Python will issue a number of errors:

```
dir_path = 'c:\user\tasks\new'
```

Error:

```
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in positior
```

Python treats \u in the path as a Unicode escape but couldn't decode it.

Now, if you escape the first backslash, you'll have other issues:

```
dir_path = 'c:\\user\tasks\new'
print(dir_path)
```

Output:

```
c:\user asks
ew
```

In this example, the `\t` is a tab and `\n` is the newline.

To make it easy, you can turn the path into a raw string like this:

```
dir_path = r'c:\user\tasks\new'
print(dir_path)
```

Convert a regular string into a raw string

To convert a regular string into a raw string, you use the built-in repr() function. For example:

```
s = '\n'
raw_string = repr(s)

print(raw_string)
```

Output:

```
'\n'
```

Note that the result raw string has the quote at the beginning and end of the string. To remove them, you can use slices:

```
s = '\n'
raw_string = repr(s)[1:-1]
print(raw_string)
```

Summary

- Prefix a literal string by the letter r or R to turn it into a raw string.
- Raw strings treat backslash a literal character.