03 Python - String Surgery; Cut & Stitch

Escape Character

```python
# Single quote
print('He said, \'Hello!\'')
# Output: He said, 'Hello!'

# Double quote
print("She exclaimed, \"That's amazing!\"")
# Output: She exclaimed, "That's amazing!"

# Tab
print('Name:\tJohn\tAge:\t32')
# Output: Name:    John    Age:    32

# Newline
print('First line\nSecond line')
# Output:
# First line
# Second line

# Backslash
print('This is a backslash: \\')
# Output: This is a backslash: \
```

1. The first `print` statement demonstrates how to use the escape character `\` before a single quote within a string to indicate that it is part of the string, rather than a delimiter for the string. The output is `He said, 'Hello!'`. Here, `\` is used to escape the single quote so that it can be printed as part of the string.

2. The second `print` statement demonstrates how to use the escape character `\` before a double quote within a string to indicate that it is part of the string, rather than a delimiter for the string. The output is `She exclaimed, "That's amazing!"`. Here, `\` is used to escape the double quote so that it can be printed as part of the string.

3. The third `print` statement demonstrates how to use the escape character `\t` within a string to insert a tab character. The output is `Name: John Age: 32`. Here, `\t` is used to insert a tab character between the words `Name:`, `John`, `Age:`, and `32`.

4. The fourth `print` statement demonstrates how to use the escape character `\n` within a string to insert a newline character.

5. The fifth `print` statement demonstrates how to use the escape character `\\` within a string to insert a backslash character. The output is `This is a backslash: \`. Here, `\\` is used to insert a backslash character between the words `This is a backslash:` and `.`.

Indexing and Slicing

```python
# indexing and slicing strings
spam = 'Hello world!'
print(spam[0]) # output: H
print(spam[4]) # output: o
print(spam[-1]) # output: !
print(spam[0:5]) # output: Hello
print(spam[:5]) # output: Hello
print(spam[6:]) # output: world!

# capturing a slice in a separate variable
fizz = spam[0:5]
print(fizz) # output: Hello

# using in and not in operators with strings
print('Hello' in 'Hello World') # output: True
print('Hello' in 'Hello') # output: True
print('HELLO' in 'Hello World') # output: False (case-sensitive)
print('' in 'spam') # output: True
print('cats' not in 'cats and dogs') # output: False
```

1. We define a string variable `spam` containing the value `'Hello world!'`.
2. We use indexing to access the first character of the string (`'H'`) and print it to the console.
3. We use indexing to access the fifth character of the string (`'o'`) and print it to the console.
4. We use negative indexing to access the last character of the string (`'!'`) and print it to the console.
5. We use slicing to capture the first five characters of the string (`'Hello'`) and print it to the console.
6. We use slicing to capture the first five characters of the string again, but this time we use the shorthand notation `[:5]`.
7. We use slicing to capture everything after the sixth character of the string (`'world!'`) and print it to the console.
8. We capture the substring `'Hello'` using slicing and store it in a separate variable `fizz`, then print it to the console.
9. We use the `in` operator to check whether the substring `'Hello'` is present in the string `'Hello World'`, which evaluates to `True`.
10. We check whether the substring `'Hello'` is present in the string `'Hello'`, which also evaluates to `True`.
11. We check whether the substring `'HELLO'` (in all capital letters) is present in the string `'Hello World'`, which is case-sensitive and evaluates to `False`.
12. We check whether an empty substring `''` is present in the string `'spam'`, which is always `True`.
13. We use the `not in` operator to check whether the substring `'cats'` is not present in the string `'cats and dogs'`, which is `False`.

String Methods

```
# Convert a string to uppercase
print('hello'.upper())  # output: HELLO

# Chain methods to convert a string to lowercase and then uppercase
print('Hello'.upper().lower())  # output: hello
print('Hello'.upper().lower().upper())  # output: HELLO

# Convert a string to lowercase
print('HELLO'.lower())  # output: hello

# Check if all characters in a string are lowercase
print('hello'.islower())  # output: True

# Check if all characters in a string are alphabetic
print('hello'.isalpha())  # output: True
print('hello123'.isalpha())  # output: False

# Check if all characters in a string are alphanumeric
print('hello'.isalnum())  # output: True
print('hello123'.isalnum())  # output: True

# Check if all characters in a string are decimal digits
print('123'.isdecimal())  # output: True

# Check if all characters in a string are whitespace
print(' '.isspace())  # output: True

# Check if a string is in title case
print('This Is Title Case'.istitle())  # output: True
print('This Is Title Case 123'.istitle())  # output: True
print('This Is not Title Case'.istitle())  # output: False
print('This Is NOT Title Case Either'.istitle())  # output: False

# Check if a string starts or ends with a particular substring
print('Hello world!'.startswith('Hello'))  # output: True
print('Hello world!'.endswith('world!'))  # output: True
```

- `upper()`: Converts all characters in a string to uppercase.
- `lower()`: Converts all characters in a string to lowercase.
- `islower()`: Returns True if all characters in a string are lowercase, False otherwise.
- `isalpha()`: Returns True if all characters in a string are alphabetic, False otherwise.
- `isalnum()`: Returns True if all characters in a string are alphanumeric, False otherwise.
- `isdecimal()`: Returns True if all characters in a string are decimal digits, False otherwise.
- `isspace()`: Returns True if all characters in a string are whitespace, False otherwise.
- `istitle()`: Returns True if a string is in title case (i.e., the first letter of each word is capitalized), False otherwise.
- `startswith(substring)`: Returns True if a string starts with the specified substring, False otherwise.
- `endswith(substring)`: Returns True if a string ends with the specified substring, False otherwise.

String Manupulation

```
# Using the join() method to concatenate a list of strings with a separator
print(', '.join(['cats', 'rats', 'bats']))   # Output: 'cats, rats, bats'

# Using the join() method with a space separator
print(' '.join(['My', 'name', 'is', 'Simon']))  # Output: 'My name is Simon'

# Using the join() method with a different separator
print('ABC'.join(['My', 'name', 'is', 'Simon']))   # Output: 'MyABCnameABCisABCSimon'

# Using the split() method to split a string into a list of substrings using space separator by default
print('My name is Simon'.split())   # Output: ['My', 'name', 'is', 'Simon']

# Using the split() method with a custom separator
print('MyABCnameABCisABCSimon'.split('ABC'))   # Output: ['My', 'name', 'is', 'Simon']

# Using the rjust() method to right-justify a string with padding spaces
print('Hello'.rjust(10))   # Output: '     Hello'
print('Hello'.rjust(20))   # Output: '               Hello'
print('Hello World'.rjust(20))   # Output: '         Hello World'

# Using the ljust() method to left-justify a string with padding spaces
print('Hello'.ljust(10))   # Output: 'Hello     '

# Using the center() method to center a string with padding spaces
print('Hello'.center(20))   # Output: '       Hello        '
print('Hello'.center(2))   # Output: 'Hello'
print('Hello'.center(20, '=')) # # Output: '=======Hello========'
```

1. `join()` method takes an iterable argument (such as a list) and concatenates its elements as a string, with the separator between each element.
2. `split()` method splits a string into a list of substrings, using the specified separator (space by default). It returns a list of substrings.
3. `rjust()` method returns a right-justified string, padded with spaces to the specified width.
4. `ljust()` method returns a left-justified string, padded with spaces to the specified width.
5. `center()` method returns a centered string, padded with spaces to the specified widt
6.

Removing Whitespace

```
spam = ' Hello World '
print(spam.strip())   # Output: 'Hello World'
print(spam.lstrip())  # Output: 'Hello World '
print(spam.rstrip())  # Output: ' Hello World'

spam = 'SpamSpamBaconSpamEggsSpamSpam'
print(spam.strip('ampS'))  # Output: 'BaconSpamEggs'
```

- strip(): removes any whitespace characters from the beginning and end of the string. If a string argument is passed, it removes any characters from the string argument that appear at the beginning or end of the string.
- lstrip(): removes any whitespace characters from the beginning of the string.
- rstrip(): removes any whitespace characters from the end of the string.
- Whitespace refers to any character that is used to separate words or other characters in a string, but that is not visible when the string is printed. Examples include spaces, tabs, and newline characters.
- In Python, the strip(), lstrip(), and rstrip() methods are commonly used to remove whitespace from strings.
- Removing whitespace is important when working with user input or when parsing text files, since extra whitespace can interfere with string comparisons and other operations.

Wiki Markup

```
import pyperclip

# Copy a string to the clipboard
```

```python
string_to_copy = "This is a test string."
pyperclip.copy(string_to_copy)

# Paste the text from the clipboard and display it
pasted_text = pyperclip.paste()
print("Pasted text:", pasted_text)
# output : Pasted text: This is a test string.
```

1. Copy and paste text from the clipboard using pyperclip.
2. Separate the lines of text and add a star to the start of each line using a loop and string concatenation.
3. Join the modified lines into a single string using the join() method.
4. Copy the new text to the clipboard using pyperclip.
5. Run the program and paste the modified text back into the desired location, such as a Wikipedia article.