String Dissection and Assembly

Contents

Today we're going to cover one of the most fundamental and useful skills in Python: manipulating strings. Think of strings as sentences or words, and we're going to learn how to be a "word mechanic" 🧑‍🔧. We'll take them apart, move the pieces around, and put them back together in new ways.

Taking Strings Apart: Slicing and Splitting

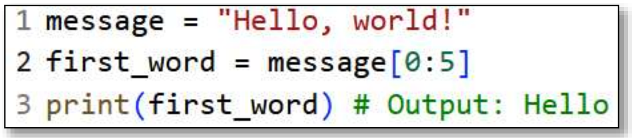
Let's learn how to break our strings down into smaller, manageable pieces. We have two main tools for this: slicing and splitting.

Slicing: Grabbing a Piece of the String

Slicing is like cutting out a specific part of a word or sentence using its character positions (called indexes). Remember, Python starts counting from 0!

Let's say we have the string message = "Hello, world!".

To get just "Hello", we need characters from index 0 up to (but not including) index 5.



```python
message = "Hello, world!"
first_word = message[0:5]
print(first_word) # Output: Hello
```

A handy shortcut: if you start from the beginning, you can leave out the first number. To get "world", we can start from index 7 and go to the end. If you go to the end, you can leave out the second number.

```python
message = "Hello, world!"
second_word = message[7:] # Start at index 7 and go to the end
print(second_word) # Output: world!
```

Splitting: Breaking a String into a List

Slicing is great, but what if you want to break a sentence into individual words? Using the **.split()** method is much easier. This tool breaks a string apart wherever it sees a specific character (a **delimiter**) and gives you a **list** of the pieces as characters.

By default, **.split()** uses spaces as the delimiter.

```python
sentence = "Learning Python is fun"
words = sentence.split(' ') # The ' ' tells it to split on spaces
print(words) # Output: ['Learning', 'Python', 'is', 'fun']
```

Notice the output is in square brackets []. This is a Python **list**, which is just an ordered collection of items. Now that we have our words in a list, we can easily access and reorder them!

You can also split by any other character. This is very useful for data that's separated by commas or dashes.

```python
data = "apple,banana,cherry"
fruits = data.split(',') # Split on the comma
print(fruits) # Output: ['apple', 'banana', 'cherry']
```

Putting Strings Back Together: Concatenation and Joining

Concatenation: The Simple Plus Sign (+)

The easiest way to combine strings is to just "add" them together with the + symbol. This is called **concatenation**.

```python
first_name = "Jane"
last_name = "Doe"
# Don't forget to add a space in between!
full_name = first_name + " " + last_name
print(full_name) # Output: Jane Doe
```

This works great for combining a few pieces, but it can get clunky if you have many parts. For that, we have a much better tool.

Joining: The Pro-Level Method

The **.join()** method is the perfect partner to **.split()**. It takes a list of strings and joins them together into a single string, using a "glue" string that you specify.

The syntax looks a little backward at first, so pay close attention:
glue.join(list_of_pieces)

Let's take the list of words we made earlier and put them back together, but this time with dashes - instead of spaces.

```python
# A list of strings
words = ['Learning', 'Python', 'is', 'fun']
# Join the list of strings into a string with - in between the words
new_sentence = "-".join(words)
print(new_sentence)
# Output: Learning-Python-is-fun
```

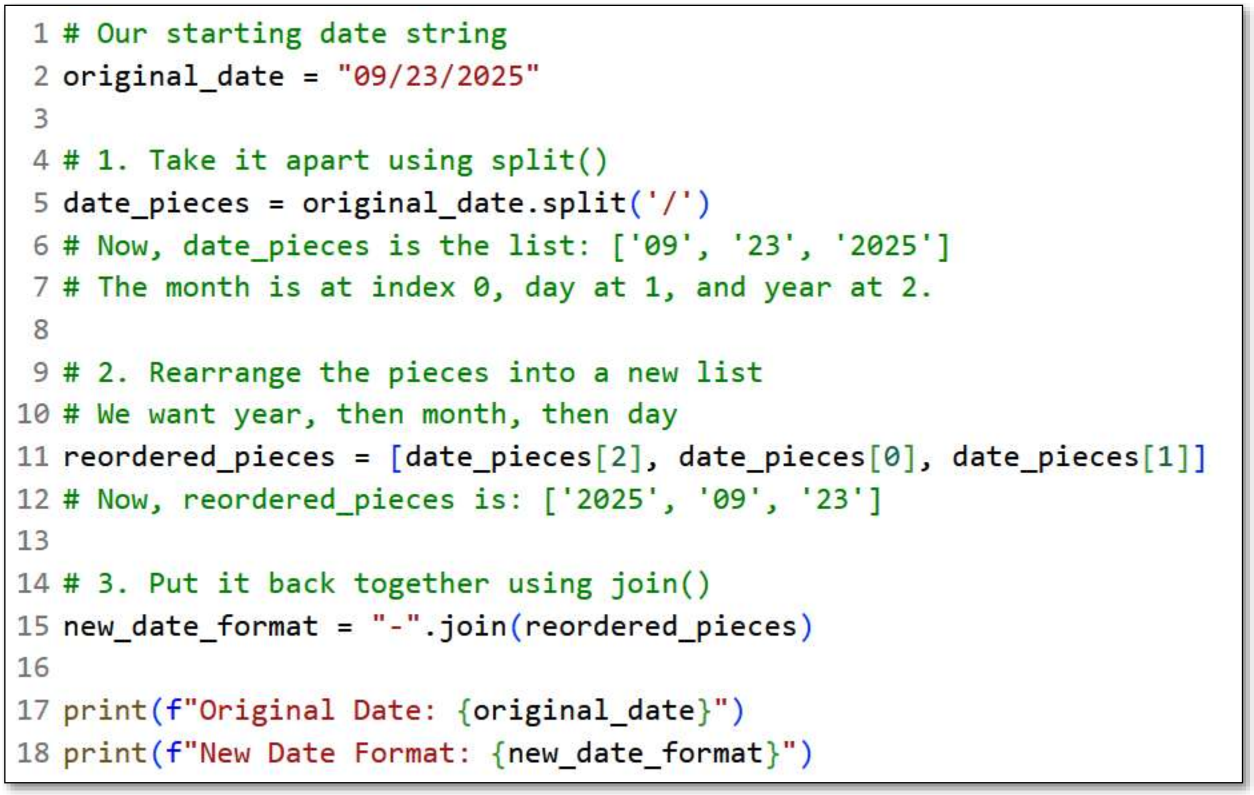
Putting It All Together: A Practical Example

Let's solve a real-world problem. Imagine you have a date in the American format MM/DD/YYYY and you need to convert it to the standard format YYYY-MM-DD.

Here's our game plan:

1. **Take it apart** at the / character.
2. **Rearrange** the pieces (year, then month, then day).
3. **Put it back together** with - as the glue.

Let's see it in action! 🎬



```python
# Our starting date string
original_date = "09/23/2025"

# 1. Take it apart using split()
date_pieces = original_date.split('/')
# Now, date_pieces is the list: ['09', '23', '2025']
# The month is at index 0, day at 1, and year at 2.

# 2. Rearrange the pieces into a new list
# We want year, then month, then day
reordered_pieces = [date_pieces[2], date_pieces[0], date_pieces[1]]
# Now, reordered_pieces is: ['2025', '09', '23']

# 3. Put it back together using join()
new_date_format = "-".join(reordered_pieces)

print(f"Original Date: {original_date}")
print(f"New Date Format: {new_date_format}")
```

There you have it! By combining **.split()** and **.join()**, you can reorder any string you want. This is an incredibly powerful technique you'll use all the time.

Practice is key, so try splitting and joining your own sentences. Happy coding! 👍

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the programs.
3. Submit in Blackboard.