CSI 1401 Week 11 Resources

by Allen Yan

This week, CSI 1401 will go through chapter 7. Compared to the monstrosity that was chapter 6, chapter 7 provides somewhat of a reprieve with its much less rigorous content. Chapter 7 will introduce some advanced string manipulation techniques in Python.

As a reminder, I will be leading a group tutoring session on CSI 1401 every Wednesday from 7:00 pm to 8:00 pm (central time). The session will be conducted online via Microsoft Meetings, where I will be available to provide interactive help to students. If the above time window does not work for you, or if you need additional help, Baylor's Tutoring Center also provides 1-on-1 online tutoring appointments. For more information on signing up for group tutoring or individual tutoring, please visit https://www.baylor.edu/tutoring.

Week 11 Important Topics

Chapter 7 – Strings

- String slicing
- Advanced string formatting
- String methods

1. String slicing

In earlier weeks, we have covered that strings in Python are structured like lists of characters, with each character accessible by its sequential index. Python uses these characteristics of strings to provide some advanced short-hand notations that allows programmers to quickly manipulate strings. These notations are called **slice notations**, and they typically come in the format of `my_str[start_idx:end_idx]`, which would "cut off" the portion of the string that starts at `start_idx` and ends at `end_idx-1`. Keep in mind that strings in Python are immutable, so the "cut-off" strings are actually new string objects copied from the original strings.

For a more concrete example of the various ways to use the slice notation, please refer to the textbook example below. Assuming `my_str` has the value of "http://en.wikipedia.org/wiki/Nasa/":

Syntax	Result	Description
`my_str[10:19]`	wikipedia	Gets the characters in indices 10-18.
`my_str[10:-5]`	wikipedia.org/wiki/	Gets the characters in indices 10-28.
`my_str[8:]`	n.wikipedia.org/wiki/Nasa/	All characters from index 8 until the end of the string.

`my_str[:23]`	http://en.wikipedia.org	Every character up to index 23, but not including my_str[23].
`my_str[:-1]`	http://en.wikipedia.org/wiki/Nasa	All but the last character.

If you are familiar with MATLAB, you may find this notation somewhat familiar. These same slicing notations can also be used on lists in Python and will effectively work the same.

2. Advanced string formatting

Up to this point, we have more or less simply outputted program results with `print()`, which sometimes could use additional formatting to make the output look prettier. One of such formatting techniques is using the **field width**. The field width specification allows the string to reserve a specified amount of character positions for a certain part of the string, and if said part of string does not fully fill the reserved positions, the remaining positions will be padded with space characters. To illustrate the syntax and effect of field width, please consider the textbook example below.

```
format_string = '{name:16}{goals:8}'

print(format_string.format(name='Player Name', goals='Goals'))
print('-' * 24)

print(format_string.format(name='Sadio Mane', goals=22))
print(format_string.format(name='Gabriel Jesus', goals=7))
```

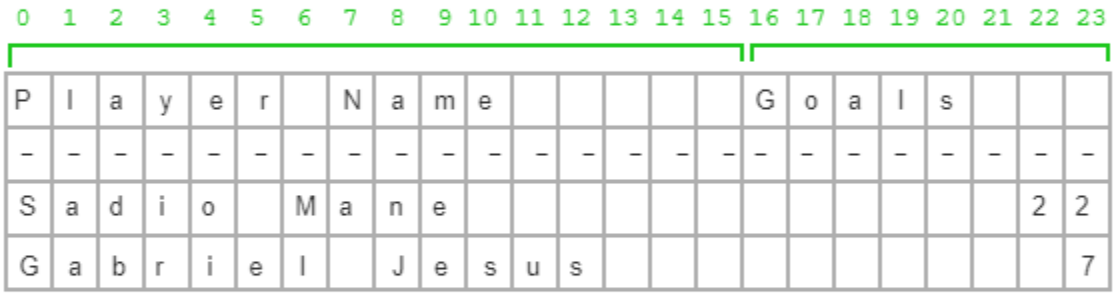

Image snipped from course zyBooks

You may also find it useful sometimes to align your output text all to a single direction, as opposed to the above example, where the first column is left-aligned, and the second column is right-aligned. To accomplish this, simply add an **alignment character** before the field width parameter. There are three alignment characters: < for left alignment, > for right alignment, and ^ for center alignment,

and of course, you may elect to use no alignment characters for Python's default alignment. Using the same example as above, the results of different alignment operations are shown below.

```
format_string = <format_string>  # Replaced in table below

print(format_string.format(name='Player Name', goals='Goals'))
print('-' * 24)

print(format_string.format(name='Sadio Mane', goals=22))
print(format_string.format(name='Gabriel Jesus', goals=7))
```

Alignment type	<format_string>	Output
Left-aligned	'{name:<16}{goals:<8}'	Player Name Goals ------------------------ Sadio Mane 22 Gabriel Jesus 7
Right-aligned	'{name:>16}{goals:>8}'	Player Name Goals ------------------------ Sadio Mane 22 Gabriel Jesus 7
Centered	'{name:^16}{goals:^8}'	Player Name Goals ------------------------ Sadio Mane 22 Gabriel Jesus 7

Image snipped from course zyBooks

Lastly, you may also specify a **fill character** before the alignment character, which will tell Python to fill the unoccupied positions of the part of string with the fill character instead of the default space character. For example:

```
format_string = '{name:#<16}{goals:$>8}'
print(format_string.format(name='Player Name', goals='Goals'))
```

will yield "Player Name#####$$$Goals".

3. String methods

The remainder of chapter 7 kind of dumps a whole bunch of available methods for strings in Python. They are meant to make your life easier when dealing with strings, but it could be stressful to memorize all of them – you don't need to, most of them are fairly straightforward in terms of naming and syntax, and you can always look them up if you forget. I will list a brief description of them below. Unless specified otherwise, all of methods listed below are invoked using the format `my_str.method_name()`.

- **`replace(old, new)`** – Returns a copy of the string with all instances of `old` replaced by `new`.
- **`replace(old, new, count)`** – Same as above, but only replaces the first `count` occurrences of `old`.
- **`find(x)` –** Returns the index of the first occurrence of x in the string, returns -1 if x is not found.
- **`find(x, start)`** – Same as above, but starts the search at index `start`.
- **`find(x, start, end)`** – Same as above, but only searches the range starting at index `start` and ends at index `end-1`.
- **`rfind(x)`** – Same as `find(x)`, but searches the string in reverse, and returns the last occurrence in the string.
- **`count(x)`** – Returns the number of times x occurs in the string.
- **`isalnum()`** – Returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.
- **`isdigit()`** – Returns True if all characters are the numbers 0-9.
- **`islower()`** – Returns True if all characters are lowercase.
- **`isupper()`** – Returns True if all characters are uppercase.
- **`isspace()`** – Returns True if all characters are whitespaces.
- **`startswith(x)`** – Returns True if string starts with x.
- **`endswith(x)`** – Returns True if string ends with x.
- **`capitalize()`** – Returns a copy of the string with the first character capitalized and the rest lowercased.
- **`lower()`** – Returns a copy of the string with all characters lowercased.
- **`upper()`** – Returns a copy of the string with all characters uppercased.
- **`strip()`** – Returns a copy of the string with leading and trailing whitespaces removed.
- **`title()`** – Returns a copy of the string as a title, with first letters of words capitalized.
- **`split()`** – Splits the string into a list of smaller strings, using whitespaces as separators, separators are not added to the list.
- **`split(sep)`** – Same as above, but using `sep` as separators.
- **`sep.join(my_list)`** – Joins the contents of `my_list` into a single string using `sep` as separators, `my_list` is a list of strings.

Useful Resources

- Basic Python Tutorial on GeeksforGeeks:
  https://www.geeksforgeeks.org/python-programming-language/
    - This page provides links to detailed explanations to many entry-level Python concepts along with examples. I encourage taking a look at it if the examples in your textbook were not clear enough.
- Official Python Documentation:
  https://docs.python.org/3/
    - This may be a bit heavy-handed for a beginner-level programmer since the official Python documentation is very thorough and technical. However, learning how to read official documentations is crucial to becoming a good programmer, because the official documentation contains information on everything you can find about Python elsewhere and more. Therefore, I encourage slowly easing yourself into learning how to read the official documentation.