Week 10

Introduction to Programming – CSI 1401

Hello and welcome to the weekly resources for CSI 1401 – Introduction to Programming 1! This week's resource will be covering the course material from week 10 in the course.

This course is designed to give you an introduction to Python, basic programming constructs, and object-oriented programming. These resources will be available every week to offer a review of the overall concepts covered in class. However, this resource should not be used to replace the textbook or your professor's lecture. They should serve as a review to help answer any general questions or provide clarity on certain topics. If you have any questions regarding CSI 1401 or need additional support, feel free to reach out to the tutoring center; but don't forget that your course instructor is the most important resource you have in this introductory programming course.

Reminder: If you have any questions about these study guides, group tutoring sessions, private 30 minute tutoring appointments, the Baylor Tutoring YouTube channel, or any tutoring services we offer, please visit our website at https://baylor.edu/tutoring or call our drop in center during open business hours (Monday-Thursday 9am – 8pm on class days at (254) 710-4135).

Keywords: String slicing, advanced formatting, string methods

---

Topic of the Week – Advanced Strings

In past chapters, we have defined strings as sequences of characters to form text. We have also mentioned that you can treat strings as containers and loop through each individual character. In this chapter, instead of thinking of strings as text, it's very important to think of them as sequences of characters.

A useful way to use strings is with string slicing. This is built on the concept that each character in the string has an index, just like elements in a list. This is done with start and end indexes. For example, `my_string[0:4]` would get the first four characters from the `my_string` variable. The start index is inclusive while the end index is exclusive. Use string slicing when you need to quickly manipulate or access part of a string.

You can omit any of the integers to go until the start or end of the string. For example, in continuing with the above example, `my_string[:4]` would produce the same output.

You can also include a third integer in the brackets, which is called a stride. This will skip over the specified number of characters in the range provided.

Lastly, setting the start index to be greater than the end index will produce an empty string, not an error!

Whenever you use a string slice, it is important to remember that Python creates a new object for that slice. This is because strings are immutable. So even if you change the original string, the slice object will remain the same!

Highlight 1 – String Formatting

We can use special characters, specifications, and other programming constructs to style the output of text in Python. These constructs are added onto the basic `print()` function by using `f` (f for fancy string!).

The first form of formatting available to you is called field width. This defines the minimum number of characters in a string. Field width is the number after the colon (`:`) in a `print()` statement. For example, `print(f"{"Baylor University:50}")` will print Baylor University with a field width of 50. Since there are not 50 characters in that string, Python will "pad" or add on extra spaces to reach 50 characters. Field width is helpful when you want to make tables or create columns since different types will have different right or left alignments by default.

Speaking of text alignment, that is another factor you can control as a programmer. The alignment is specified before the field width integer. `<` is for left, `>` is for right, and `^` is for centered. The text will be aligned as specified within the field width. Continuing with the above example, `print(f"{"Baylor University":^50}")` will align Baylor University in the middle of the added spaces.

Lastly, you can also decide what character Python adds when it needs to pad a string to reach the field width. This is specified before the alignment character. For example, `print(f"{"Baylor University":-^50}")` would place Baylor University in the center of repeated dashes.

Highlight 2 – Methods

The final highlight from this part of the chapter is string methods. These are meant to be helpful when manipulating strings and provide helpful shortcuts. zyBooks does a great job in providing tables and lists of these methods with a quick overview of what they do. Although you don't need to memorize all of the methods, it would be helpful to give them a quick overview before the mastery exam to ensure you can get easy points.

- `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.

Check Your Learning

1. Write a formatted string to be right aligned, filled with 0s, and have a 16 field width.
2. How do you get an empty string using string slicing?
3. What is the output of this code?

```
my_str = "Banana chocolate chip muffins"
new_str = my_str[8:]
new_new_str = new_str[:3]
print(new_new_str)
```

Things Students May Struggle With

1. String are immutable, meaning a new object is created whenever you want to modify a string.
2. Treat string slicing like list elements because they behave very similarly.
3. The order for string formatting is fill character, text alignment, and field width.
4. Remember that methods are implemented using dot notation.

Thank you for using this weekly resource! Don't forget to check out our website for group tutoring times, video tutorials, and lots of other course resources: https://baylor.edu/tutoring. The answers to the "Check Your Learning" questions are below.

Answers

1. `f'{"example string":0>16}'`
2. The start index must be greater than the end index in the brackets or the indexes must be equal.
3. `hoc`
   Remember that the end index is not included in the string slice.
   ```
   new_str = "hocolate chip muffins"
   new_new_str = "hoc"
   ```