Study Guide: Strings

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz. The string readings in this section are great syntax guides to help you on the Strings Practice Quiz.

In the Strings segment, you learned about the parts of a string, string indexing and slicing, creating new strings, string methods and operations, and formatting strings.

Knowledge

String Operations and Methods

- **.format()** - String method that can be used to concatenate and format strings.
    - **{:.2f}** - Within the .format() method, limits a floating point variable to 2 decimal places. The number of decimal places can be customized.
- **len(string)** - String operation that returns the length of the string.
- **string[x]** - String operation that accesses the character at index [x] of the string, where indexing starts at zero.
- **string[x:y]** - String operation that accesses a substring starting at index [x] and ending at index [y-1]. If x is omitted, its value defaults to 0. If y is omitted, the value will default to len(string).
- **string.replace(old, new)** - String method that returns a new string where all occurrences of an old substring have been replaced by a new substring.
- **string.lower()** - String method that returns a copy of the string with all lowercase characters.

Coding skills

Skill Group 1

- Use a **for** loop to iterate through each letter of a string.
- Add a character to the front of a string.
- Add a character to the end of a string.
- Use the **.lower()** string method to convert the case (uppercase/lowercase) of the letters within a string variable. *This method is often used to eliminate cases as a factor when comparing two strings. For example, all lowercase "cat" is not equal to "Cat" because "Cat" contains an uppercase letter. To be able to compare the two strings to see if they are the same word, you can use the .lower() string method to remove capitalization as a factor in the string comparison.*

```
# This function accepts a given string and checks each character of
# the string to see if it is a letter or not. If the character is a
# letter, that letter is added to the end of the string variable
# "forwards" and to the beginning of the string variable "backwards".
def mirrored_string(my_string):

    # Two variables are initialized as string data types using empty
    # quotes. The variable "forwards" will hold the "my_string"
    # minus any character that is not a letter. The "backwards"
    # variable will hold the same letters as "forwards", but in
    # in reverse order.
    forwards = ""
    backwards = ""

    # The for loop iterates through each character of the "my_string"
    for character in my_string:

        # The if-statement checks if the character is not a space.
        if character.isalpha():

            # If True, the body of the loop adds the character to the
            # to the end of "forwards" and to the front of
            # "backwards".
            forwards += character
            backwards = character + backwards

        # If False (meaning the character is not a letter), no action
        # is needed. This coding approach results prevents any
        # non-alphabetical characters from being written to the
        # "forwards" and "backwards" variables. The for loop will
        # restart until all characters in "my_string" have been
        # processed.

    # The final if-statement compares the "forwards" and "backwards"
    # strings to see if the letters are the same both forwards and
    # backwards. Since Python is case sensitive, the two strings will
    # need to be converted to use the same case for this comparison.
    if forwards.lower() == backwards.lower():
        return True
    return False
```

Run

Reset

Skill Group 2

- Use the **format()** method, with **{}** placeholders for variable data, to create a new string.
- Use a formatting expression, like **{:.2f}**, to format a float variable and configure the number of decimal places to display for the float.

```
# This function converts measurement equivalents. Output is formatted
# as, "x ounces equals y pounds", with y limited to 2 decimal places.
def convert_weight(ounces):

    # Conversion formula: 1 pound = 16 ounces
    pounds = ounces/16

    # The result is composed using the .format() method. There are two
    # placeholders in the string: the first is for the "ounces"
    # variable and the second is for the "pounds" variable. The second
    # placeholder formats the float result of the conversion
    # calculation to be limited to 2 decimal places.
    result = "{} ounces equals {:.2f} pounds".format(ounces,pounds)
    return result


print(convert_weight(12)) # Should be: 12 ounces equals 0.75 pounds
print(convert_weight(50.5)) # Should be: 50.5 ounces equals 3.16 pounds
print(convert_weight(16)) # Should be: 16 ounces equals 1.00 pounds
```

Run

Reset

Skill Group 3

- Within the **format()** parameters, select characters at specific index [ ] positions from a variable string.
- Use the **format()** method, with **{}** placeholders for variable data, to create a new string.

```
# This function generates a username using the first 3 letters of a
# user's last name plus their birth year.
def username(last_name, birth_year):

    # The .format() method will use the first 3 letters at index
    # positions [0,1,2] of the "last_name" variable for the first
    # {} placeholder. The second {} placeholder concatenates the user's
    #  "birth_year" to that string to form a new string username.
    return("{}{}".format(last_name[0:3],birth_year))


print(username("Ivanov", "1985"))
# Should display "Iva1985"
print(username("Rodriguez", "2000"))
# Should display "Rod2000"
print(username("Deng", "1991"))
# Should display "Den1991"

```

Run

Reset

Skill Group 4

- Use the **.replace()** method to replace part of a string.
- Use the **len()** function to get the number of index positions in a string.
- Slice a string at a specific index position.

```
# This function checks a given schedule entry for an old date and, if
# found, the function replaces it with a new date.
def replace_date(schedule, old_date, new_date):

    # Check if the given "old_date" appears at the end of the given
    # string variable "schedule".
    if schedule.endswith(old_date):

        # If True, the body of the if-block will run. The variable "n" is
        # used to hold the slicing index position. The len() function
        # is used to measure the length of the string "new_date".
        p = len(old_date)

        # The "new_schedule" string holds the updated string with the
        # old date replaced by the new date. The schedule[:-p] part of
        # the code trims the "old_date" substring from "schedule"
        # starting at the final index position (or right-side) counting
        # towards the left the same number of index positions as
        # calculated from len(old_date). Then, the code schedule[-p:]
        # starts the indexing position at the slot where the first
        # character of the "old_date" used to be positioned. The
        # .replace(old_date, new_date) code inserts the "new_date" into
        # the position where the "old_date" used to exist.
        new_schedule = schedule[:-p] + schedule[-p:].replace(old_date, new_date)

        # Returns the schedule with the new date.
        return new_schedule

    # If the schedule does not end with the old date, then return the
    # original sentence without any modifications.
    return schedule


print(replace_date("Last year's annual report will be released in March 2023", "2023", "2024"))
# Should display "Last year's annual report will be released in March 2024"
print(replace_date("In April, the CEO will hold a conference", "April", "May"))
# Should display "In April, the CEO will hold a conference"
print(replace_date("The convention is scheduled for October", "October", "June"))
# Should display "The convention is scheduled for June"

```

Run

Reset

Python practice information

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

- Welcome to Python
- Online Python Interpreter