HyperionDev

String Handling

Task

Visit our website

Introduction

One of the most crucial concepts to grasp when it comes to programming is string handling. You will need to be very comfortable with string handling, so it is important to refresh and consolidate your knowledge. In this task, you will briefly recap some key points and then learn to create more advanced programs with strings which use more functions and programming techniques.

Indexing strings

You can think of the string "`Hello world!`" as a list and each character in the string as an item with a corresponding index.

```
' H e l l o   w o r l d ! '
  0 1 2 3 4 5 6 7 8 9 10 11
```

The space and the exclamation point are included in the character count, so "Hello world!" is 12-characters long, from "H" at index zero to "!" at index 11.

```
String = "Hello"
print(String[0]) # H
print(String[1]) # e
print(String[2]) # l
print(String[3]) # l
print(String[4]) # o
```

If you specify an index, you will get the character at that position in the string. You can also slice strings by specifying a range from one index to another; remember that the character at the starting index is included and that the character at the ending index is not.

Note: Slicing a string does not modify the original string. You can capture a slice from one variable in a separate variable. Try running the following code in your IDE or an interactive shell:

```
# String slicing by extracting a substring from the original string
original_string = "Hello world!"
new_string = original_string[0:5]
print(new_string)
print(original_string)
```

What will be printed by the code above? By slicing and storing the resulting substring in another variable, you can have both the whole string and the substring handy for quick, easy access.

String methods

Once you understand strings and their indexing, the next step is to master using some of the common string methods. These are built-in modules of code that perform certain operations on strings. These methods save time since there is no need to write the code over and over again to perform certain operations. The most common string methods are (where `s` is the variable that contains the string we are working with):

- `s.lower()` and `s.upper()` – respectively converts a string to either lowercase or uppercase.
- `s.strip()` – removes any whitespace from the beginning and end of a string.
- `s.strip(‘,’)` – optional input to `s.strip()`. The string provided as input will be removed from the beginning and end of the target string. In this case, the “`,`” character gets removed from the `s` string.
- `s.find(‘text’)` – searches for a specific text and returns its position in the string you are searching. If the string is not found, `-1` is returned.
- `s.replace('oldText', 'newText')` – replaces any occurrence of `‘oldText’` with `‘newText’`.
- `s.split('word')` – breaks down a string into a list of smaller pieces. The string is separated based on what is called a **delimiter**. This is a string or char value that is passed to the method. If no value is given, it will automatically split the string using whitespace as the delimiter and create a list of the characters.
- `“”.join(string_list)` – takes a list of strings or characters and joins them to create one string. You can specify the character, if any, you wish to use to join the list elements. For example, `“@”.join([“apples”, “bananas”, “carrots”])` would output “`apples@bananas@carrots`”.

Examine the **example code file** to see how some of these methods can be used.

Escape characters

Python uses the backslash (\) as an escape character. The backslash (\) is used as a marker character to tell the compiler/interpreter that the next character has some special meaning. The backslash, together with certain other characters, is known as an escape sequence. Some useful escape sequences are listed below:

- \n – Newline
- \t – Tab

The escape character can also be used if you need to include quotation marks within a string. Placing a backslash (\) before a quotation mark prevents it from terminating the string prematurely. What would the code below print out? Try it and see!

```
print("Hello \n\"Bob\"")
```

You can also put a backslash in front of another backslash to include a backslash in a string.

```
print("The escape sequence \\n creates a new line in a print statement")
# Output: The escape sequence \n creates a new line in a print statement
```

String building

As you know, the best practice is to create strings through `.format()` and, by extension, f-strings. For example:

```
name = "Peter"
print("Hello, {}!".format(name))
```

```
name = "Peter"
print(f"Hello, {name}!")
```

However, sometimes a form of concatenation is needed in order to build a string. One of the main reasons for writing a program is to manipulate information. We turn meaningless data (e.g. 24) into useful information (e.g. Tom is 24 years old). String building allows us to put data in a format that turns data into information. This is important for working with text files and databases, and when you send data from the back end to the front end.

Below is an example of string building using a `while loop`.

```
number_builder = ""
i = 0

# Loop until i reaches 50
while i <= 50:
    if i % 2 == 0:
        # Add an even number and a space
        number_builder += str(i) + " "
    i += 1
print(number_builder)
```

Here, every time `i` is even, it gets cast as a string and added to the `number_builder` string, which starts off empty – (“ ”) – until i is greater than or equal to `50`.

Another way to do this is with the `.join()` method you have just learnt about. As mentioned, this method takes a list and joins the elements together to make a string. We can rewrite the above example as below to incorporate `.join()`:

```
# Note the variable has to be a list rather than a string
number_builder = []
i = 0

while i <= 50:
    if i % 2 == 0:
        # Add the even number as a string to the list
        number_builder.append(str(i))
    i += 1
# Join the list elements into a single string with spaces and print
print(" ".join(number_builder))
```

Here, we have made `number_builder` a list, and for each iteration of the loop, an even number gets appended to the list using **`append()`**. Finally, in the print statement, the elements are all joined together with a space in between. You may have noticed that `i` is cast to a string before being appended. This is because you cannot make an integer act like a string without casting it – only strings and characters can be joined together.

For both examples, the output would be:

```
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
```

Instructions

Read and run the accompanying **example files** provided before doing the task, in order to become more comfortable with the concepts covered in this task.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task

Follow these steps:

1. Create a file called **alternative.py**
2. Write a program that prompts the user to enter a string and makes each alternate character into an uppercase character and each other alternate character a lowercase character.

   e.g. The string “Hello World” would become “HeLlO WoRlD”

- Now, try starting with the same string but making each alternative word lowercase and uppercase.

  e.g. The string “I am learning to code” would become “i AM learning TO code”.

Tip: Using the `split()` and `join()` methods will help you here.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.
