Exemplar_ Strings

January 7, 2024

Exemplar: Work with strings in Python

0.1 Introduction

Data professionals work with a lot of string data. For example, while analyzing the results of a marketing campaign, you may need to review item descriptions or customer names, which is stored as string data. Becoming comfortable working with strings in Python is essential for the work of a data professional.

In this lab, you'll practice coding in Python and working with strings. You'll work with a store ID, ZIP Code, and a custom URL for the store you're gathering data on.

0.2 Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace this with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the `[Double-click to record your response.]` with your own answers.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

0.3 Task 1: Check and change data types

Now that you have experience in marketing, you've moved on to market research. Your new task is collecting store data for future analysis. In this task, you're given a four-digit numeric store ID stored in a variable called `store_id`.

1. Convert `store_id` to a string and store the result in the same variable.
2. Confirm the type of `store_id` after the conversion.

[1]:
```
store_id = 1101

# 1. ### YOUR CODE HERE ###
```

```
store_id = str(store_id)

# 2. ### YOUR CODE HERE ###
print(type(store_id))
```

```
<class 'str'>
```

Hint 1

You can refer back to what you learned about changing data types.

Hint 2

When using Python's built-in functions such as `type()` and `str()`, make sure to pass the variables or data you're converting into the parenthesis.

Hint 3

Use the `str()` function in Python to convert the initial value of the `store_id` variable into a string.

To reassign `store_id` as the same variable, write `store_id`, then the `=` operator, then `str(store_id)` on a new line in the cell.

0.4 Task 2: String concatenation

As you continue gathering data, you realize that the `store_id` variable is actually the ZIP Code where the store is located, but the leading `0` has been cut off.

- Define a function called `zip_checker` that accepts the following argument:
  - `zipcode` - a string with either four or five characters
- Return:
  - If `zipcode` has five characters, and the first two characters are NOT `'00'`, return `zipcode` as a string. Otherwise, return `'Invalid ZIP Code.'`. (ZIP Codes do not begin with 00 in the mainland U.S.)
  - If `zipcode` has four characters and the first character is NOT `'0'`, the function must add a zero to the beginning of the string and return the five-character `zipcode` as a string.
  - If `zipcode` has four characters and the first character is `'0'`, the function must return `'Invalid ZIP Code.'`.

Example:

```
 [IN] zip_checker('02806')
[OUT] '02806'

 [IN] zip_checker('2806')
[OUT] '02806'

 [IN] zip_checker('0280')
[OUT] 'Invalid ZIP Code.'

 [IN] zip_checker('00280')
```

```
[OUT] 'Invalid ZIP Code.'
```

Note that there is more than one way to solve this problem.

[2]:
```
### YOUR CODE HERE ###
def zip_checker(zipcode):
    if len(zipcode) == 5:
        if zipcode[0:2] =='00':
            return 'Invalid ZIP Code.'
        else:
            return zipcode
    elif zipcode[0] != '0':
        zipcode = '0' + zipcode
        return zipcode
    else:
        return 'Invalid ZIP Code.'
```

Hint 1

- Refer to what you learned about defining functions in the "Define functions and returning values" video.
- Refer to what you learned about `if, elif, else` statements in the "Use if, elif, else statements to make decisions" video.

Hint 2

- Use the `len()` function to determine the length of `zipcode`.
- One way to concatenate strings is to use the `+` operator.

Hint 3

One approach is to check the length of `zipcode`. If it's five characters long, check if the first two (indices `[0:2]`) are `'00'`. If they are, return `'Invalid ZIP Code.'`. Otherwise, return `zipcode`. Or else if the first character is not `'0'`, concatenate `'0'` and `zipcode`, then return `zipcode`. Otherwise, return `Invalid ZIP Code.`

0.4.1 Test your function

Test your function against the following cases by running the cell below.

[3]:
```
print(zip_checker('02806'))     # Should return 02806.
print(zip_checker('2806'))      # Should return 02806.
print(zip_checker('0280'))      # Should return 'Invalid ZIP Code.'
print(zip_checker('00280'))     # Should return 'Invalid ZIP Code.'
```

```
02806
02806
Invalid ZIP Code.
Invalid ZIP Code.
```

0.5 Task 3: Extract the store ID

Now imagine that you've been provided `url`, which is a URL containing the store's actual ID at the end of it.

1. Extract the seven-character store ID from the end of `url` and assign the result to a variable called `id`.
2. Print the contents of `id`.

[4]:
```
url = "https://exampleURL1.com/r626c36"

# 1. ### YOUR CODE HERE ###
id = url[-7:]

# 2. ### YOUR CODE HERE ###
print(id)
```

```
r626c36
```

Hint 1

Refer back to what you learned about saving strings to variables and string slicing.

Hint 2

To slice a string that is stored in a variable, use indexing brackets on the variable. Inside the brackets, specify the starting and stopping values of the array, separated by a colon (`:`).

Hint 3

- Note that the store ID `r626c36` is seven characters long. In the brackets containing the string indices, `[-7:]` indicates to extract the last seven characters of the string. There is no stop value in this case because an empty stop value implies `len(string)` (the ending index of the string).
- Use the `print()` function to display the slice.

0.6 Task 4: String extraction function

You have many URLs that contain store IDs, but many of them are invalid—either because they use an invalid protocol (the beginning of the URL) or because the store ID is not seven characters long.

- The correct URL protocol is `https:` Anything else is invalid.
- A valid store ID must have exactly seven characters.

Define a function called `url_checker` that accepts the following argument: * `url` - a URL string

Return: * If both the protocol and the store ID are invalid: * print two lines: `'{protocol} is an invalid protocol.'` `'{store_id} is an invalid store ID.'` * If only the protocol is invalid: * print: `'{protocol} is an invalid protocol.'` * If only the store ID is invalid: * print: `'{store_id} is an invalid store ID.'` * If both the protocol and the store ID are valid, return the store ID.

In the above cases, `{protocol}` is a string of the protocol and `{store_id}` is a string of the store ID.

Example:

```
 [IN] url_checker('http://exampleURL1.com/r626c3')
[OUT] 'http: is an invalid protocol.'
      'r626c3 is an invalid store ID.'

 [IN] url_checker('ftps://exampleURL1.com/r626c36')
[OUT] 'ftps: is an invalid protocol.'

 [IN] url_checker('https://exampleURL1.com/r626c3')
[OUT] 'r626c3 is an invalid store ID.'

 [IN] url_checker('https://exampleURL1.com/r626c36')
[OUT] 'r626c36'
```

Note that there is more than one way to solve this problem.

[5]:
```python
# Sample valid URL for reference while writing your function:
url = 'https://exampleURL1.com/r626c36'

### YOUR CODE HERE ###
def url_checker(url):
    url = url.split('/')
    protocol = url[0]
    store_id = url[-1]
    # If both protocol and store_id bad
    if protocol != 'https:' and len(store_id) != 7:
        print(f'{protocol} is an invalid protocol.',
            f'\n{store_id} is an invalid store ID.')
    # If just protocol bad
    elif protocol != 'https:':
        print(f'{protocol} is an invalid protocol.')
    # If just store_id bad
    elif len(store_id) != 7:
        print(f'{store_id} is an invalid store ID.')
    # If all ok
    else:
        return store_id
```

Hint 1

- Consider using the `split()` method to split `url` into separate elements, then assign the first element and last element to their own variables, so you can check them. Refer to the str.split() Python documentation for more information on this method.

Hint 2

There are different possible outcomes, so the logic should follow an if, elif, elif, else branching

scheme.

Hint 3

Use string formatting to insert the URL-specific protocol string and store ID string into your print statements.

0.6.1 Test your function

Test your function against the following cases by running the cell below.

[6]:
```
                                                    # Should return:
url_checker('http://exampleURL1.com/r626c3')        # 'http: is an invalid protocol.
 ↪'
print()                                             # 'r626c3 is an invalid store␣
 ↪ID.'

url_checker('ftps://exampleURL1.com/r626c36')       # 'ftps: is an invalid protocol.
print()
url_checker('https://exampleURL1.com/r626c3')       # 'r626c3 is an invalid store␣
 ↪ID.'
print()
url_checker('https://exampleURL1.com/r626c36')      # 'r626c36'
```

```
http: is an invalid protocol.
r626c3 is an invalid store ID.

ftps: is an invalid protocol.

r626c3 is an invalid store ID.
```

[6]:
```
'r626c36'
```

0.7 Conclusions

What are your key takeaways from this lab?

- Strings are instrumental in storing important data, such as unique identifiers.
- String concatenation helps you combine data that is stored in different strings.
- String formatting is useful when inserting specific values into reusable string templates.
- Functions boost efficiency by reusing code for repeated tasks.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.