Cleaning and Preparing Data in Python: Takeaways

Syntax

TRANSFORMING AND CLEANING STRINGS

- Replace a substring within a string:

```
green_ball = "red ball".replace("red", "green")
```

- Remove a substring:

```
friend_removed = "hello there friend!".replace(" friend", "")
```

- Remove a series of characters from a string:

```
bad_chars = ["'", ",", ".", "!"]
string = "We'll remove apostrophes, commas, periods, and exclamation marks!"
for char in bad_chars:
    string = string.replace(char, "")
```

- Convert a string to title cases:

```
Hello = "hello".title()
```

- Check a string for the existence of a substring:

```
if "car" in "carpet":
    print("The substring was found.")
else:
    print("The substring was not found.")
```

- Split a string into a list of strings:

```
split_on_dash = "1980-12-08".split("-")
```

- Slice characters from a string by position:

```
last_five_chars = "This is a long string."[:5]
```

- Concatenate strings:

```
superman = "Clark" + " " + "Kent"
```

STRING FORMATTING AND FORMAT SPECIFICATIONS

- Insert values into a string in order:

```
continents = "France is in {} and China is in {}".format("Europe", "Asia")
```

- Insert values into a string by position:

```
squares = "{0} times {0} equals {1}".format(3,9)
```

- Insert values into a string by name:

```
population = "{name}'s population is {pop} million".format(name="Brazil", pop=209)
```

- Format specification for precision of two decimal places:

```
two_decimal_places = "I own {:.2f}% of the company".format(32.5548651132)
```

- Format specification for comma separator:

```
india_pop = The approximate population of {} is {}".format("India",1324000000)
```

- Order for format specification when using precision and comma separator:

```
balance_string = "Your bank balance is {:,.2f}"].format(12345.678)
```

Concepts

- When working with comma separated value (CSV) data in Python, it's common to have your data in a "list of lists" format, where each item of the internal lists are strings.
- If you have numeric data stored as strings, sometimes you will need to remove and replace certain characters before you can convert the strings to numeric types, like `int` and `float`.
- Strings in Python are made from the same underlying data type as lists, which means you can index and slice specific characters from strings like you can lists.
- The `str.format()` method allows you to insert values into strings without explicitly converting them.
- The `str.format()` method also accepts optional format specifications, which you can use to format values so they are easier to read.

Resources

- Python Documentation: String Methods
- Python Documentation: Format Specifications
- PyFormat: Python String Formatting Reference