Working with Strings 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 different cases:

```
HELLO = "hello".upper()

hello = "HELLO".lower()

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

- Strip whitespace from the beginning and end of a string:

```
clean_name = "   Margaret Hamilton   ".strip()
```

- 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"
```

- Join a list of strings into a single string:

```
sentence = " ".join(['These', 'are', 'the', 'words', 'in', 'the', 'sentence.'])
```

CATCHING EXCEPTIONS

- Catch an exception:

```
try:
    func_that_may_generate_exception()
except ValueError: # if you don't specify the exception
                   # type, all exceptions will be caught
    run_this_func_only_if_valueerror()
```

STRING FORMATTING AND FORMAT SPECIFICATIONS

- Insert a value intro a string:

```
world_cup_semifinal = "The final score was Croatia {}, England {}".format(2,1)
```

- Format specification for 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)
```

- Format specification for padding and alignment:

```
left_aligned_padded_20_chars   = "| {:<20} |".format("MY_VALUE")
center_aligned_padded_20_chars = "| {:^20} |".format("MY_VALUE")
right_aligned_padded_20_chars  = "| {:>20} |".format("MY_VALUE")
```

- Format specification to derive padding width from a variable:

```
left_aligned  = "| {:<{}} |".format("MY_VALUE", padding_width)
```

- Order for format specification:

```
balance_string = "Your bank balance is {:>20,.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.
- In Python, errors are technically called exceptions. If an exception is raised, it will stop your program executing.
- When an exception is raised, a traceback is displayed which shows context from the part of the code which generated the error.
- If an exception is raised from within a function, the traceback will include context from multiple places so you can trace the error back through your code.
- In order in order to handle these exceptions, we use try/except.
    - Try/Except consists of a `try` block and an `except` block
    - The code in the `try` block will always be executed
    - The code in the `except` block will only be executed if the `try` code raises an exception
    - You can specify the specific type of error you want to catch, however if you don't, the try/except will catch every exception type
- 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 more easily read.

Resources

- [Python Documentation: String Methods]()
- [Python Documentation: Format Specifications]()
- [PyFormat: Python String Formatting Reference]()