Manipulating Strings

Multiple ways to type strings

Double quotes
Useful because using single quotes can create issues with apostrophes. Double quotes make things like this work:

```
>>> spam = "That is Alice's cat."
```

Escape characters
These let you use characters that are otherwise impossible in a string. They're characterized by a backslash (\) followed by the character that you want to add to a string.

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Escape character	Prints as
\'	Single quote
\"	Double quote
\t	Tab
\n	Newline (line break)
\\	Backslash

Raw strings ignore escape characters and print backslashes in the string itself.

Triple quotes / multiline strings - everything inside is considered to be a string. See catnapping.py for example.

Multiline comments - see chapter 5 (https://automatetheboringstuff.com/chapter6/)

Indexing and slicing strings

“Strings use indexes and slices the same way lists do. 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
```

spam[0] = H
spam[0:5] = Hello
spam[-1] = !

In / not operators - see chapter

Useful string methods - upper(), lower(), isupper(), and islower()

Upper() returns all letters in a string to uppercase
Lower() returns all letters in a string to lowercase

isuupper() returns a boolean value based on whether-or-not the values are uppercase
islower() returns a boolean value based on whether-or-not the values are lowercase

isX string methods

```
isalpha() returns True if the string consists only of letters and is
not blank.

isalnum() returns True if the string consists only of letters and
numbers and is not blank.

isdecimal() returns True if the string consists only of numeric
characters and is not blank.

isspace() returns True if the string consists only of spaces, tabs,
and new-lines and is not blank.

istitle() returns True if the string consists only of words that
begin with an uppercase letter followed by only lowercase letters.
```

startswith() and endswith() methods - both are self explanatory. They evaluate whether-or-not a string starts or ends with the given input.

```
>>> 'Hello world!'.startswith('Hello')
True
>>> 'Hello world!'.endswith('world!')
True
```

join() and split() string methods

join() concatenates multiple strings

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
```

split() splits multiple strings

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

A common use of split() is to split a multiline string along the newline characters

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment".

Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a
container in the
fridge', 'that is labeled "Milk Experiment".', '', 'Please do not
drink it.',
'Sincerely,', 'Bob']
```

Justifying text with rjust(), ljust(), and center()

```
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello     '
```

A second argument can justify a fill character like so:

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

Removing Whitespace with strip(), rstrip(), and lstrip()

These methods strip whitespace from the designated section of the string

```
>>> spam = '    Hello World     '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World '
>>> spam.rstrip()
'    Hello World'
```

“Optionally, a string argument will specify which characters on the ends should be stripped”

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Copying and pasting strings from the clipboard with the pyperclip module

“The pyperclip module has copy() and paste() functions that can send text to and receive text from your computer’s clipboard. Sending the output of your program to the clipboard will make it easy to paste it to an email, word processor, or some other software.

Pyperclip does not come with Python. To install it, follow the directions for installing third-party modules in Appendix A. After installing the pyperclip module, enter the following into the interactive shell:

```
>>> import pyperclip
>>> pyperclip.copy('Hello world!')
>>> pyperclip.paste()
'Hello world!'
```

Of course, if something outside of your program changes the clipboard contents, the paste() function will return it. For example, if I copied this sentence to the clipboard and then called paste(), it would look like this:

```
>>> pyperclip.paste()
'For example, if I copied this sentence to the clipboard and then
called
paste(), it would look like this:'
```

See chapter text for projects