

Work with files in Python

You previously explored how to open files in Python as well as how to read them and write to them. You also examined how to adjust the structure of file contents through the `.split()` method. In this reading, you'll review the `.split()` method, and you'll also learn an additional method that can help you work with file contents.

Parsing

Part of working with files involves structuring its contents to meet your needs. **Parsing** is the process of converting data into a more readable format. Data may need to become more readable in a couple of different ways. First, certain parts of your Python code may require modification into a specific format. By converting data into this format, you enable Python to process it in a specific way. Second, programmers need to read and interpret the results of their code, and parsing can also make the data more readable for them.

Methods that can help you parse your data include `.split()` and `.join()`.

.split()

The basics of .split()

The `.split()` method converts a string into a list. It separates the string based on a specified character that's passed into `.split()` as an argument.

In the following example, the usernames in the `approved_users` string are separated by a comma. For this reason, a string containing the comma (",") is passed into `.split()` in order to parse it into a list. Run this code and analyze the different contents of `approved_users` before and after the `.split()` method is applied to it:

```
approved_users = "elarson,bmoreno,tshah,sgilmore,eraab"
print("before .split():", approved_users)
approved_users = approved_users.split(",")
print("after .split():", approved_users)
```

```
before .split(): elarson,bmoreno,tshah,sgilmore,eraab
after .split(): ['elarson', 'bmoreno', 'tshah', 'sgilmore', 'eraab']
```

Before the `.split()` method is applied to `approved_users`, it contains a string, but after it is applied, this string is converted to a list.

If you do not pass an argument into `.split()`, it will separate the string every time it encounters a whitespace.

Note: A variety of characters are considered whitespaces by Python. These characters include spaces between characters, returns for new lines, and others.

The following example demonstrates how a string of usernames that are separated by space can be split into a list through the `.split()` method:

```
removed_users = "wjaffrey jsoto abernard jhill awilliam"
print("before .split():", removed_users)
removed_users = removed_users.split()
print("after .split():", removed_users)
```

```
before .split(): wjaffrey jsoto abernard jhill awilliam
after .split(): ['wjaffrey', 'jsoto', 'abernard', 'jhill', 'awilliam']
```

Because an argument isn't passed into `.split()`, Python splits the `removed_users` string at each space when separating it into a list.

Applying `.split()` to files

The `.split()` method allows you to work with file content as a list after you've converted it to a string through the `.read()` method. This is useful in a variety of ways. For example, if you want to iterate through the file contents in a `for` loop, this can be easily done when it's converted into a list.

The following code opens the `"update_log.txt"` file. It then reads all of the file contents into the `updates` variable as a string and splits the string in the `updates` variable into a list by creating a new element at each whitespace:

```
with open("update_log.txt", "r") as file:
    updates = file.read()
updates = updates.split()
```

After this, through the `updates` variable, you can work with the contents of the `"update_log.txt"` file in parts of your code that require it to be structured as a list.

Note: Because the line that contains `.split()` is not indented as part of the `with` statement, the file closes first. Closing a file as soon as it is no longer needed helps maintain code readability. Once a file is read into the `updates` variable, it is not needed and can be closed.

.join()

The basics of .join()

If you need to convert a list into a string, there is also a method for that. The `.join()` method concatenates the elements of an iterable into a string. The syntax used with `.join()` is distinct from the syntax used with `.split()` and other methods that you've worked with, such as `.index()`.

In methods like `.split()` or `.index()`, you append the method to the string or list that you're working with and then pass in other arguments. For example, the code `usernames.index(2)`, appends the `.index()` method to the variable `usernames`, which contains a list. It passes in 2 as the argument to indicate which element to return.

However, with `.join()`, you must pass the list that you want to concatenate into a string in as an argument. You append `.join()` to a character that you want to separate each element with once they are joined into a string.

For example, in the following code, the `approved_users` variable contains a list. If you want to join that list into a string and separate each element with a comma, you can use `",".join(approved_users)`. Run the code and examine what it returns:

```
approved_users = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab"]
print("before .join():", approved_users)
approved_users = ",".join(approved_users)
print("after .join():", approved_users)
```

```
before .join(): ['elarson', 'bmoreno', 'tshah', 'sgilmore', 'eraab']
after .join(): elarson,bmoreno,tshah,sgilmore,eraab
```

Before `.join()` is applied, `approved_users` is a list of five elements. After it is applied, it is a string with each username separated by a comma.

Note: Another way to separate elements when using the `.join()` method is to use `"\n"`, which is the newline character. The `"\n"` character indicates to separate the elements by placing them on new lines.

Applying .join() to files

When working with files, it may also be necessary to convert its contents back into a string. For example, you may want to use the `.write()` method. The `.write()` method writes string data to a file. This means that if you have converted a file's contents into a list while working with it, you'll need to convert it back into a string before using `.write()`. You can use the `.join()` method for this.

You already examined how `.split()` could be applied to the contents of the `"update_log.txt"` file once it is converted into a string through `.read()` and stored as `updates`:

```
with open("update_log.txt", "r") as file:
    updates = file.read()
updates = updates.split()
```

After you're through performing operations using the list in the `updates` variable, you might want to replace `"update_log.txt"` with the new contents. To do so, you need to first convert updates back into a string using `.join()`. Then, you can open the file using a `with` statement and use the `.write()` method to write the `updates` string to the file:

```
updates = " ".join(updates)
with open("update_log.txt", "w") as file:
    file.write(updates)
```

The code `" ".join(updates)` indicates to separate each of the list elements in `updates` with a space once joined back into a string. And because `"w"` is specified as the second argument of `open()`, Python will overwrite the contents of `"update_log.txt"` with the string currently in the `updates` variable.

Key takeaways

An important element of working with files is being able to parse the data it contains. Parsing means converting the data into a readable format. The `.split()` and `.join()` methods are both useful for parsing data. The `.split()` method allows you to convert a string into a list, and the `.join()` method allows you to convert a list into a string.