BYU IDAHO CSE 110 | Programming Building Blocks

11 Prepare: Preparation Material

Overview

To this point, most of the programs you have written get input from the user, then take actions based on that input, and display something to the screen. Your programs can become even more useful and meaningful when they can work with other information, potentially huge amounts of data, that has already been stored somewhere (in other words, it doesn't always need to come solely from user input). For example, consider a health department that tracks incidence of a certain disease, an animation company that works with the scenes of the movie, or a web store that maintains an inventory of users, products, and orders.

Depending on the context, the information in the examples above may be stored in various formats or database systems, but at the most fundamental level, they are stored as files on a computer or a server. You are likely familiar with using files on your computer such as images, word processing documents, spreadsheets, etc. In normal use, you will open these files and read sections, find important information, make decisions, update the document, etc.

As you've learned in previous lessons, by writing programs, we can have the computer perform tasks over and over again, making decisions and taking actions along the way. This ability matches up very nicely with a file that contains lots of data. The file may contain more information than one person could internalize or make decisions about, but a program could sift

through the large amount of information and take actions or bring things to the attention of the user.

In this lesson you will learn how to write programs that work with files. You'll learn how to uncover important information, such as companies with rising stock prices, or countries with disease outbreaks.

Preparation Material

Watch the following videos:

- Opening and reading from files (9 mins)
- Splitting strings (4 mins)
- Removing whitespace from strings (4 mins)
- Putting it together: Reading files and parsing strings (8 mins)

I. OPENING FILES

As shown in the videos, a file can be opened in Python with the following syntax:

```
open(filename)
```

It's important to close the file after you are finished. A best practice in Python is to use the **`with`** syntax so that it will automatically close the file for you when your program leaves that block of code:

```
with open(filename) as file_variable:
    # Code to do something with the file goes here
```

If the file you were reading from was called **`colors.txt`** and you wanted it to be saved into a variable **`color_file`**, you would use the

following:

```
with open("colors.txt") as color_file:
    # Code to do something with the color file goes here
```

Please note, the **open** function has other options that you won't use in this course, such as the ability to create new files, or make changes to existing files, etc. You could search the internet to learn more about these topics if you are interested.

II. READING DATA FROM FILES

There are several ways to read data from files in Python. One of the most straightforward approaches is to use a **for** loop to iterate over each line, the same way you would if you were iterating through the items in a list.

Once the file has been opened, you can iterate through each line as follows:

```
with open("colors.txt") as color_file:

    for line in color_file:
        # Code to do something with each line goes here
```

III. PARSING STRINGS

Once you have loaded the data from a line in the file into a variable, you may need to clean up that line or break it up into individual pieces to use separately.

Removing extra whitespace

One common thing to do is to remove any extra space at the beginning or end of a string. This "space" can be in the form of regular spaces, tab characters, or newline characters (**\n**). In

Python, there is a built-in function named **`.strip()`** that strips these characters off of the string. It can be used as follows:

```
clean_line = old_line.strip()
```

Please note that the **`.strip()`** function, does not change the original string variable, but instead it generates and returns a new one that you can save into a variable.

Splitting a string into pieces

Another common task with strings is to want to separate them into individual pieces. Python contains a built-in function **`.split()`** that can split a string up based on whatever character you define that separates the pieces.

For example, if you wanted to split up a string using a space as the separator so that each word would be a separate variable, you could use the following:

```
sentence = "I will go and do"

words = sentence.split(" ")
# The variable "words" is now a list that contains each word.

# You can iterate through each word and do something with it, such as display it:
for word in words:
    print(word)
```

If you have another character that separates the pieces of your strings, such as a comma, you could split on that using the code **`.split(",")`** .