Hyperiondev

TASK

Working with External Data Sources — Input

Visit our website

Introduction

Welcome to the Input Task!

Until now, the Python code you've been writing has only received input in one manner and has only displayed output in one way — you type input using the keyboard and its results are displayed on the console. But what if you want to read information from a file on your computer and write that information to another file? This process is called file I/O (the "I/O" stands for "input/output") and Python has some built-in functions that handle this for you.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A note from the Hyperion Team

The Python community is alive and growing at a staggering rate. An active community has many benefits. New people bring new ideas, fresh perspectives and different levels of experience.

One such benefit is the proliferation of packages. If you need functionality, there is a good chance that someone else needed it before you. The beauty of the Python community is that most of these functions become packages.

"There should be one — and preferably only one — obvious way to do it."
-The Zen of Python

This is not always the case. Sometimes you have options, many options. Feel free to explore them. **Here** are five more packages you should know about:

Working with files in Python

Files are an important source of information in Python. Before moving on, we want to make sure you know how to read the most simple type of files, text files (.txt), using Python.

Python includes a built-in file type. This is a bit like a string data structure, but much more complex.

The line below creates a file object named *f* that is linked to the **example.txt** file in this folder. You'll learn about objects in a later task.

```
f = open('example.txt', 'r+')
```

We are using a built-in function called *open* and passing it the arguments **example.txt** and *r+*. The code in the example above means that *f* is open for reading. The first argument (**example.txt**) is the name of the file that is to be read.

The second argument is the mode in which the file will be read. Some of the possible values for the mode are:

Mode	Description
r	Opens file for reading only
w	Opens file for writing only.
r+	Opens file for both reading and writing.

Here we intend to read and write from/to a text file named **example.txt**, which is already in the same folder as this file. Python will look in this directory for **example.txt**, and try to read its contents.

Reading files in Python

There are two common ways of reading files in Python:

1. The most common way to read from a file is to **loop over the lines of the file**. We can directly loop over the variable *f*, which is stored in Python as a list of lines — each line is one line of the file.

```
for line in f:
	print("The first character of this line is: " + line[0] +
"\n")
	print("The entire line is: " + line)
```

Always close files when done with them by using *f.close()* to free up the resources it was using. Notice this is a function that takes zero arguments.

We could build up all the lines of the text file into one large string called *contents* as follows:

```
contents = ""
f = open('example.txt', 'r+') # Open the file again!

for line in f:
	contents = contents + line

f.close() # Always close files when done with them.
```

We now have the contents of an external resource (a text file) stored inside our program in a variable called *contents*. That's pretty powerful! But for now, let's print the contents to a screen:

```
print(contents)
```

2. Another way of reading a file is using the *read()* method. The syntax for this method is as follows:

```
f = open('example.txt','r+')
newContents = f.read()
```

The code in the example above reads the entire file (**example.txt**). You could also pass an integer argument to the *read()* method. This argument would specify the number of characters to read from the file.

File encoding in Python

An extra optional argument can be passed to the *open()* function. This argument specifies the encoding of the file. The official documentation for Python explains that "serialising a string into a sequence of bytes is known as encoding, and recreating the string from the sequence of bytes is known as decoding".

There are several different methods used to decode or encode the file. The optional argument that is passed to the *open()* function specifies which of these methods to use. This should only be used in text mode. The default encoding is platform-dependent (whatever *locale.getpreferredencoding()* returns). Using the default encoding could result in some strange characters being displayed when Python reads a file. For example:

```
['ï»¿This is an example file!\n', "You've read from it! Congrats!"]
```

We can correct this by specifying an encoding method, as shown below:

```
f = open('example.txt','r+', encoding='utf-8')
```

See the codecs module for the list of supported encodings.

A note from our coding mentor
Joseph

I'd like to tell you about how Apple introduced the Macintosh to the world. They did it with the biggest and most expensive advertising platform available: the 1984 Super Bowl. The advert played on the theme of totalitarianism in George Orwell's book 1984. Apple made a sneaky reference to overcoming IBM by conveying the power of personal computing found in a Macintosh by a depiction of the destruction of "Big Brother".

The Macintosh was the first successful mouse-driven computer with a graphical user interface and was based on the Motorola 68000 microprocessor. Its price was $2 500. The Macintosh came with various applications as part of the package. These included MacPaint, which made use of the mouse and MacWrite, which demonstrated WYSIWYG (What You See Is What You Get) word processing.

Instructions

First, read **example.py**, open it using IDLE.

- **example.py** should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of **example.py** and try your best to understand.

- You may run **example.py** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task

Write a program that reads the data from the text file called **DOB.txt** and prints it out in two different sections in the format displayed below:

Name
A Masinga
Etc.

Birthdate
21 July 1988
Etc.

Optional Bonus Task

- Create a new Python file in this folder called **optional_task.py**.

- Create a new text file in this folder called **input.txt**. In the **input.txt** file enter some text, making sure it is at least a few lines long.

- Write a program the will count the number of characters, words and lines in the file.

- Your program should also count the total number of vowels (a's, e's, i's, o's and u's) in the file.

- Print out your results.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.