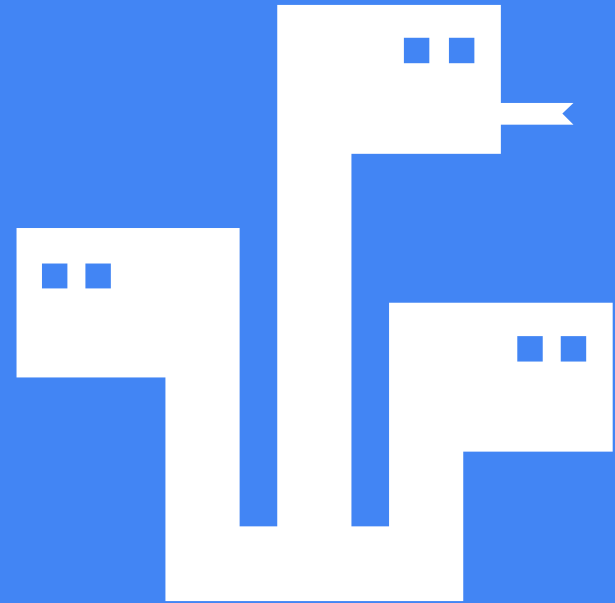


File Input/Output

Python - Nick Reynolds
April 28, 2017



File Writing

File Input/Output



Open Function

- Opens a file by a path
- The path can be relative, so list.txt means “next to this Python file”

New File Object

File Path

Mode

```
shop = open("list.txt", "w")
shop.write("Apple\n")
shop.write("Orange\n")
shop.close()
```

Modes:

- **"r"**
Read
- **"r+"**
Read/Write
- **"w"**
Write (new file if not exists)
- **"w+"**
Read/Write (new file if not exists)
- **"a"**
Append Write
- **"a+"**
Append Read/Write

Write Function

- Writes a string to the file
- `\n` is used as the new line character in text editing

Write this

```
shop = open("list.txt", "w")
shop.write("Apple\n")
shop.write("Orange\n")
shop.close()
```

“Write this *string* to this *file object*”

Close Function

- Tells the system you're done with the file
- Python makes sure it writes everything to the file and closes it

```
shop = open("list.txt", "w")  
shop.write("Apple\n")  
shop.write("Orange\n")  
shop.close()
```

With as Syntax

- We can simplify the previous piece of code with a context handler

```
shop = open("list.txt", "w")
shop.write("Apple\n")
shop.write("Orange\n")
shop.close()
```

Context handlers will automatically close the file for you.

Code in the handler is indented.

```
with open("list.txt", "w") as shop:
    shop.write("Apple\n")
    shop.write("Orange\n")
```

File Reading

File Input/Output



Read Function

- Read gets the entire contents of a file

```
with open("list.txt", "r") as shop:  
    print(shopping.read())
```

.read() returns a string, which we can assign to a variable, and use just like any other string.

```
with open("list.txt", "r") as shop:  
    x = shopping.read()  
    print(x[1])  
    if "Orange" in x:  
        print("yes")
```

.read() puts the entire file in memory, which may not be desirable

Line by Line

- Either use a for loop around the file object
- Or `readline()`

```
with open("list.txt", "r") as shop:  
    for line in shop:  
        print(line)
```

Sometimes you may want to read lines not in a loop, e.g. if you want to get the header row of a file. So we use `.readline()`

```
with open("list.txt", "r") as shop:  
    heading = shop.readline()  
    print(heading)  
    for line in shop:  
        print(line)
```

We saved the first line in the heading variable, to use later.

Pen and Paper

Checkpoint



Using CSVs

File Input/Output

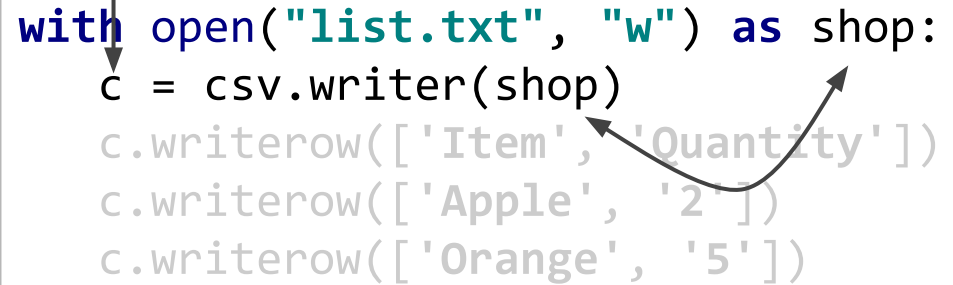


Writing to a CSV

- We first make a CSV writer for our file object

Our CSV Writer

```
with open("list.txt", "w") as shop:  
    c = csv.writer(shop)  
    c.writerow(['Item', 'Quantity'])  
    c.writerow(['Apple', '2'])  
    c.writerow(['Orange', '5'])
```



We have now set up **c** so that we can use it to interact with the file

Writing to a CSV

- Writerow() is used to add to the file

```
with open("list.txt", "w") as shop:  
    c = csv.writer(shop)  
    c.writerow(['Item', 'Quantity'])  
    c.writerow(['Apple', '2'])  
    c.writerow(['Orange', '5'])
```

The writerow function takes in a list e.g.
 ['Item', 'Quantity']
And writes these as separate columns.

The list.txt file now looks like this:

Item,Quantity Apple,2 Orange,5	Item	Quantity
	Apple	2
	Orange	5

Text

As a table in excel

Reading a CSV

- Reader reads the file in as a list of lists
- DictReader reads the file as a list of dicts with the headers as keys

```
with open("list.txt", "r") as shop:  
    read = csv.reader(shop)  
    for row in read:  
        print(row)
```

Prints

```
['Item', 'Quantity']  
['Apple', '2']  
['Orange', '5']
```

```
with open("list.txt", "r") as shop:  
    read = csv.DictReader(shop)  
    for row in read:  
        print(row)
```

Prints

```
{'Quantity': '2', 'Item': 'Apple'}  
{'Quantity': '5', 'Item': 'Orange'}
```

Practical



Practical & Assignment 2



References

- <https://docs.python.org/3/tutorial/inputoutput.html>
- <https://thenounproject.com/>