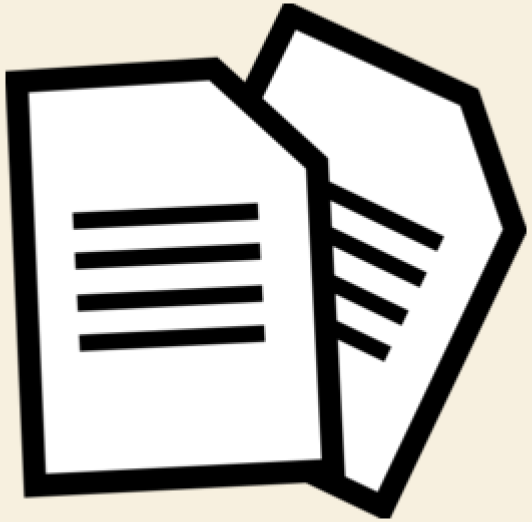
FILES

Slides by Björgvin B. Björgvinsson

FILES

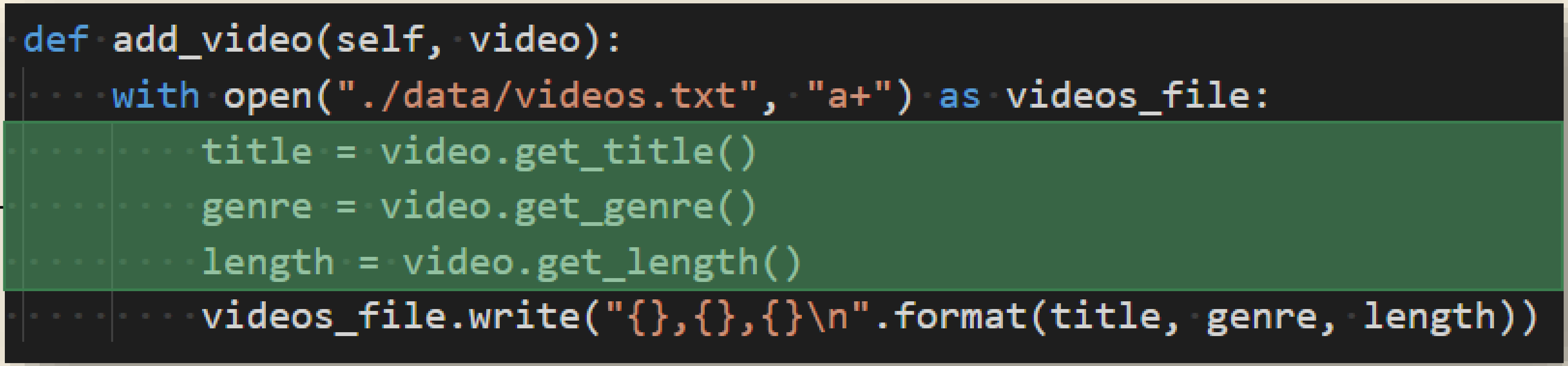
- There are many ways to structure data
  - You can use many types of file formats
  - txt files and csv files are very common
  - But how do we store class instances in files?
- Lets see some examples



FILES

- One way is to store each class instance in a single line where each attribute is separeted by a delimeter
- This way we will need to construct a string from the class instance attributes and write them to a file
  - The more instance attributes a class has the more prone to error this method becomes

This will work but it obvious that if the class had any more instance attributes the code could become more error prone

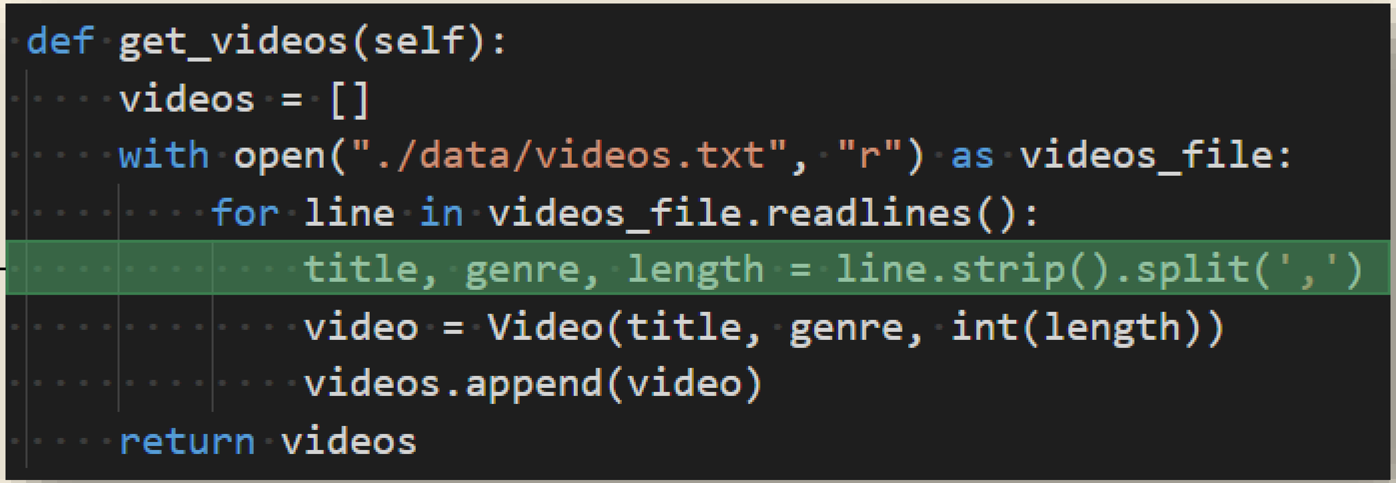


```python
def add_video(self, video):
    with open("./data/videos.txt", "a+") as videos_file:
        title = video.get_title()
        genre = video.get_genre()
        length = video.get_length()
        videos_file.write("{},{},{}\n".format(title, genre, length))
```

FILES

- Here is how we read from the file using this method to store the data
  - We will need to construct a new instance from every line in the file
  - The same goes here, the more instance attributes a class has the more prone to error this method becomes

First we strip each line of leading and trailing white spaces along with new line characters and then we split the string on the delimeter.
In the next line we construct a Video object and supply it with the data from the line we are currently reading

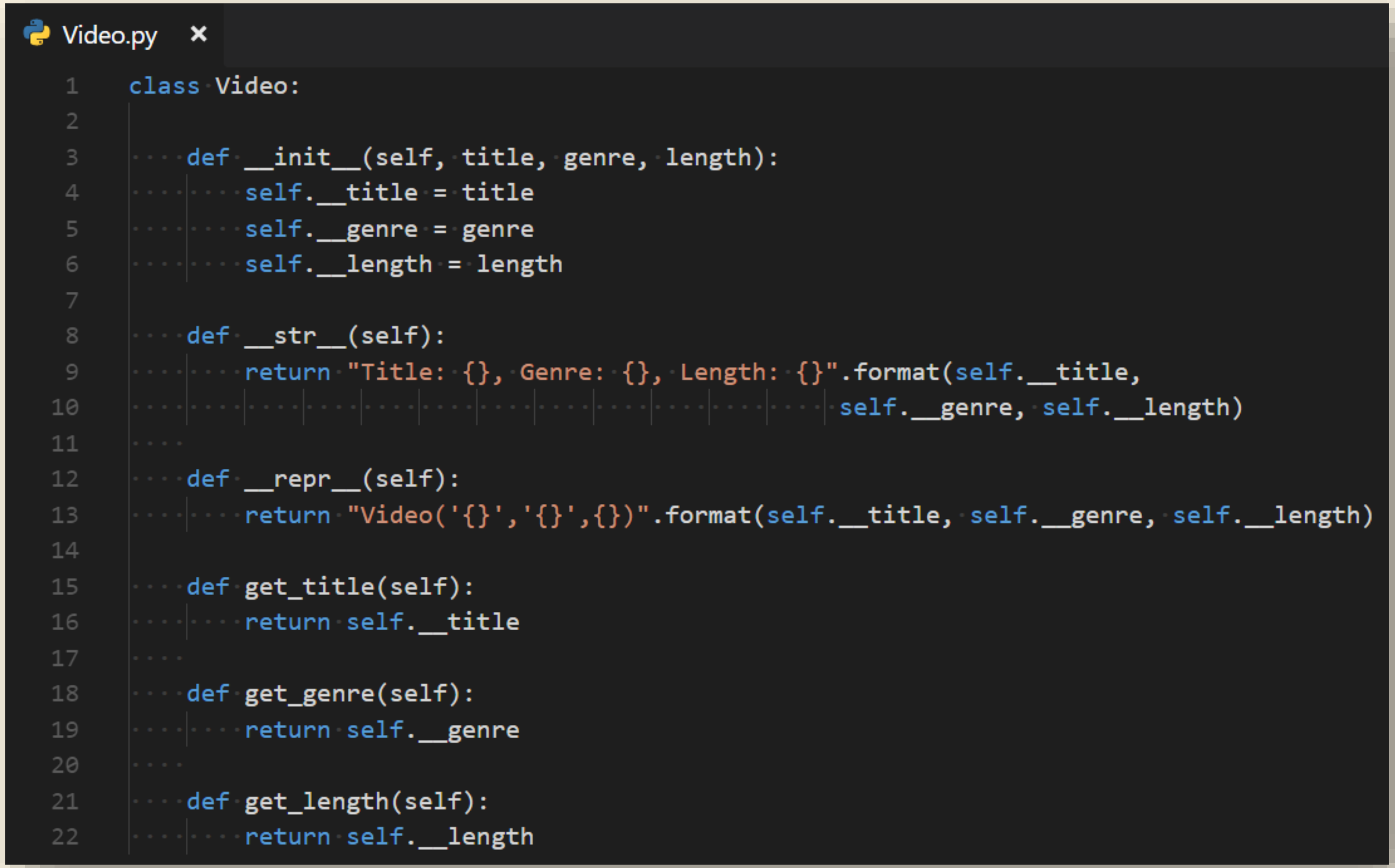


```python
def get_videos(self):
    videos = []
    with open("./data/videos.txt", "r") as videos_file:
        for line in videos_file.readlines():
            title, genre, length = line.strip().split(',')
            video = Video(title, genre, int(length))
            videos.append(video)
    return videos
```

Slides by Björgvin B. Björgvinsson

FILES

- Another way would to make use of the __repr__() method
  - We can implement the __repr__() method so it returns a string that represents the construction of an instance



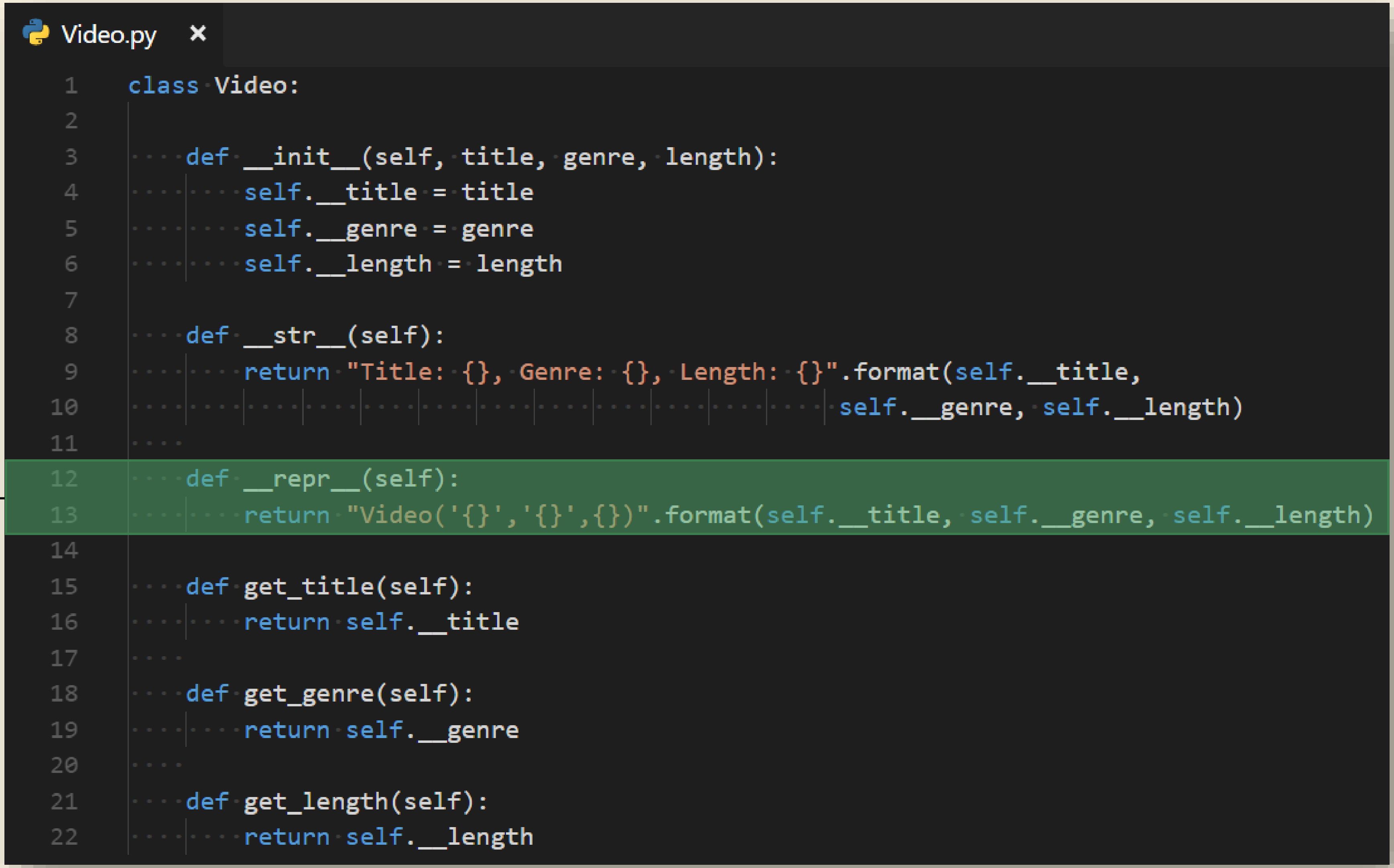
Video.py

```python
class Video:

    def __init__(self, title, genre, length):
        self.__title = title
        self.__genre = genre
        self.__length = length

    def __str__(self):
        return "Title: {}, Genre: {}, Length: {}".format(self.__title,
                                                          self.__genre, self.__length)

    def __repr__(self):
        return "Video('{}','{}',{})".format(self.__title, self.__genre, self.__length)

    def get_title(self):
        return self.__title

    def get_genre(self):
        return self.__genre

    def get_length(self):
        return self.__length
```

Slides by Björgvin B. Björgvinsson

FILES

Lets say that a particular Video instance has the **title: Blade runner,**
genre: Sci-fi
length: 200
That means that the __repr__ method would return the string: Video('Blade runner', 'Sci-fi', 200) which we could pass to the eval function!

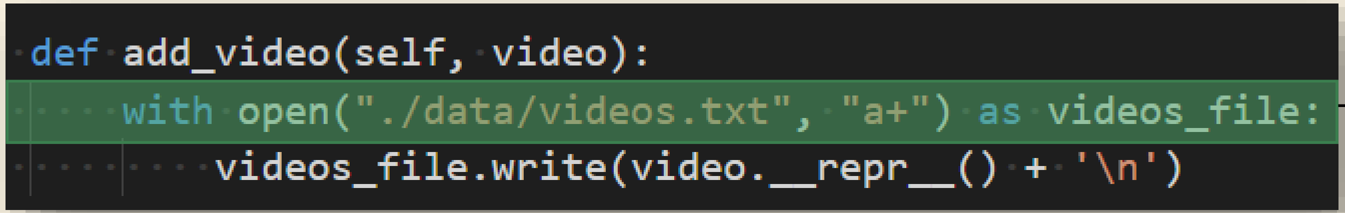


Video.py

```python
class Video:

    def __init__(self, title, genre, length):
        self.__title = title
        self.__genre = genre
        self.__length = length

    def __str__(self):
        return "Title: {}, Genre: {}, Length: {}".format(self.__title,
                                                         self.__genre, self.__length)

    def __repr__(self):
        return "Video('{}','{}',{})".format(self.__title, self.__genre, self.__length)

    def get_title(self):
        return self.__title

    def get_genre(self):
        return self.__genre

    def get_length(self):
        return self.__length
```

FILES

- By using the __repr__() method we can change the add_video function
  - Now it is much easier to read and less lines of code!



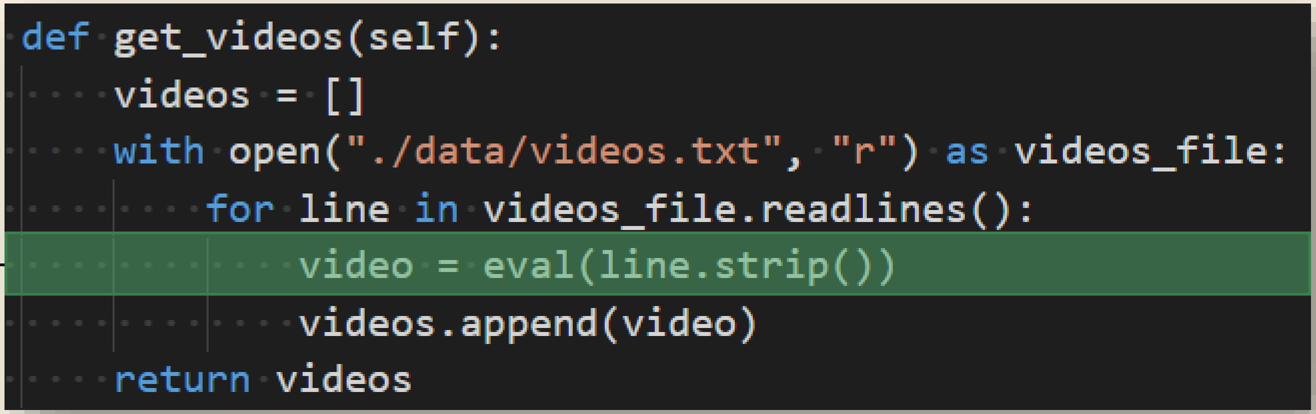
```python
def add_video(self, video):
    with open("./data/videos.txt", "a+") as videos_file:
        videos_file.write(video.__repr__() + '\n')
```

We add the newline charcter so that each video is on its own line in the text file

FILES

- Now the get_videos method looks like this
  - Even though it is not shorter, if we were to add instance attributes to the Video class we would not need to change this method, only the string returned by __repr__() in the Video class

Lets say that a particular Video instance has the following attributes:
title: Blade runner,
genre: Sci-fi
length: 200
That means that the __repr__ method would return the string:
Video('Blade runner', 'Sci-fi', 200)
which we could pass to the eval function!



```python
def get_videos(self):
    videos = []
    with open("./data/videos.txt", "r") as videos_file:
        for line in videos_file.readlines():
            video = eval(line.strip())
            videos.append(video)
    return videos
```

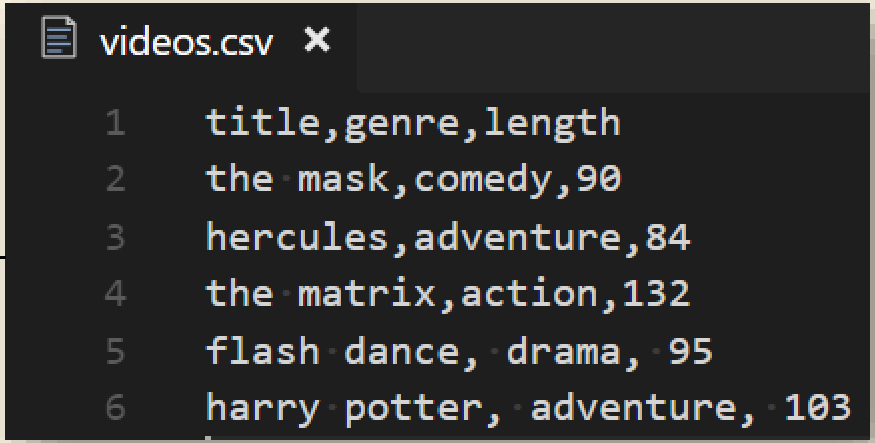
FILES

- With that said, it is worth mentioning that using eval is usually considered bad practice
  - Here are a few reasons:
    - There is almost always a better way to do it
    - It is very dangerous and insecure
    - Makes debugging difficult
    - It is slow

FILES

- It is quite common to work with csv files and Python has a built in module called csv that makes working with csv files easier

Consider this csv file. Notice that the first line contains the name of the fields. This can come in very handy!

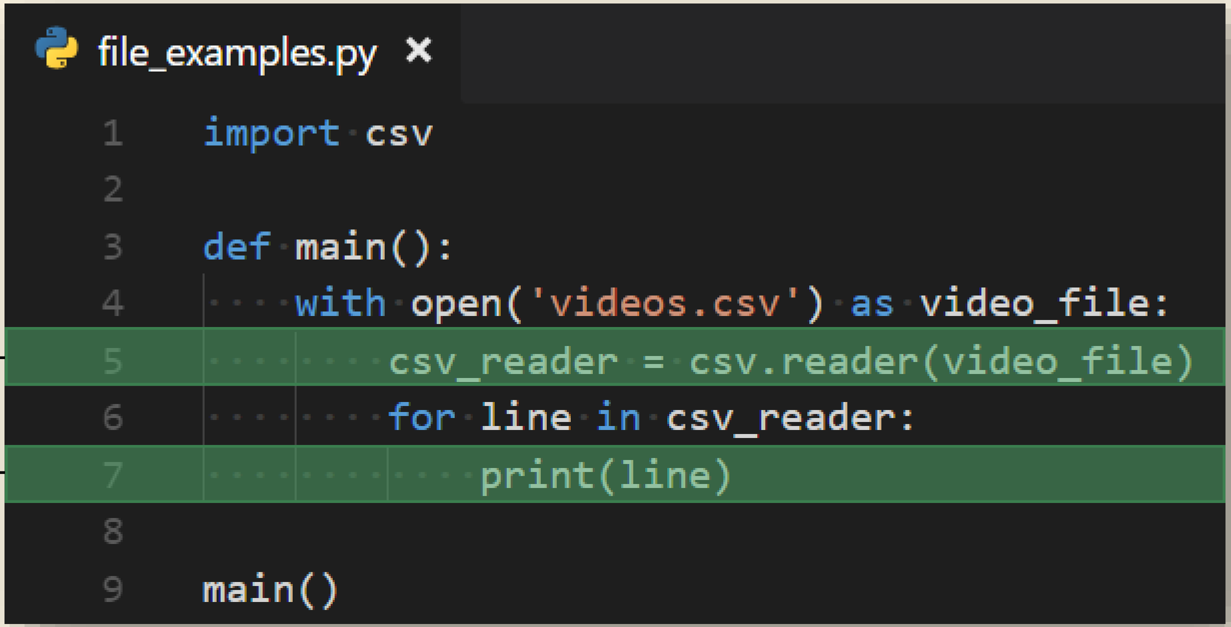


```
title,genre,length
the mask,comedy,90
hercules,adventure,84
the matrix,action,132
flash dance, drama, 95
harry potter, adventure, 103
```

FILES

We can create a csv reader like this. Do note that it can take an argument that represents a delimeter. If no argument is given it will use a comma as a delimeter

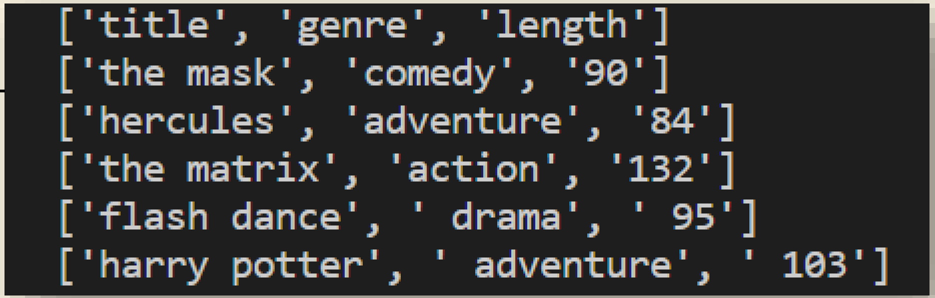
By default each line of the csv file is represented as a list the the csv reader is used



file_examples.py

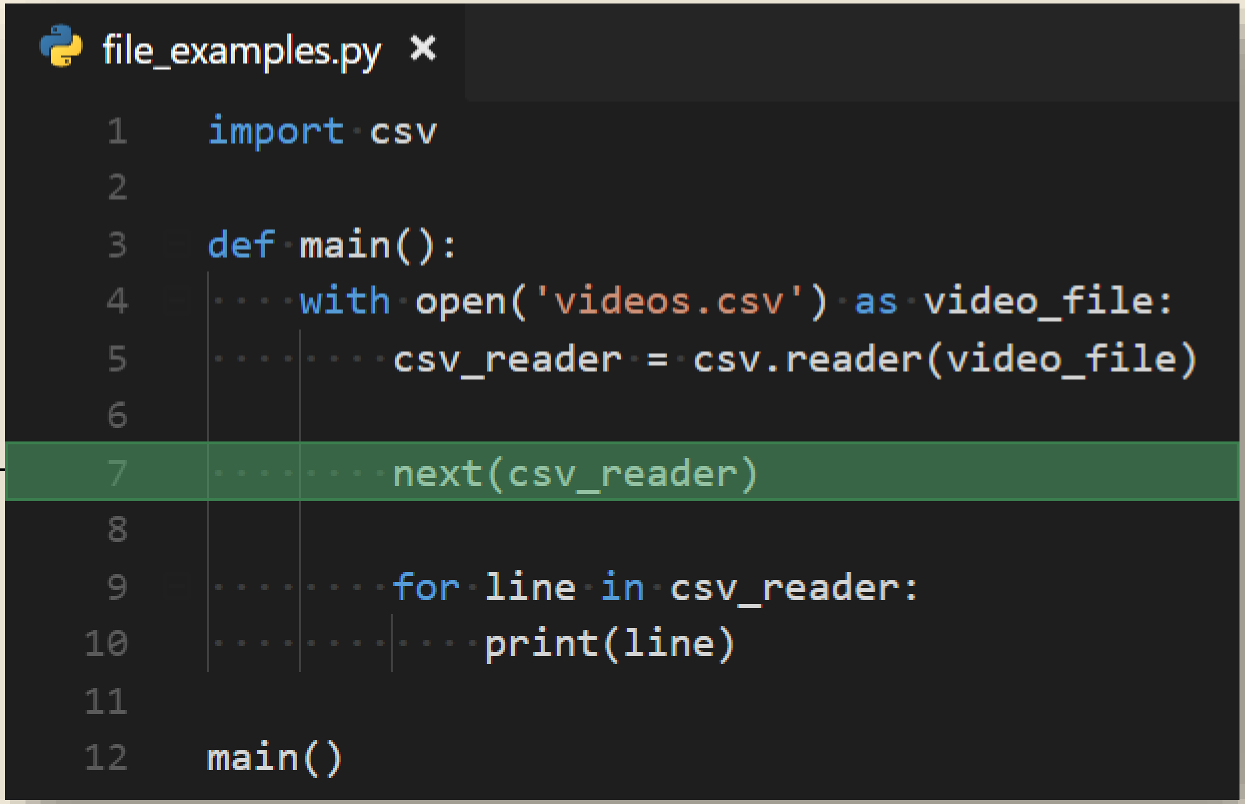
```python
import csv

def main():
    with open('videos.csv') as video_file:
        csv_reader = csv.reader(video_file)
        for line in csv_reader:
            print(line)

main()
```

This is the output



```
['title', 'genre', 'length']
['the mask', 'comedy', '90']
['hercules', 'adventure', '84']
['the matrix', 'action', '132']
['flash dance', ' drama', ' 95']
['harry potter', ' adventure', ' 103']
```

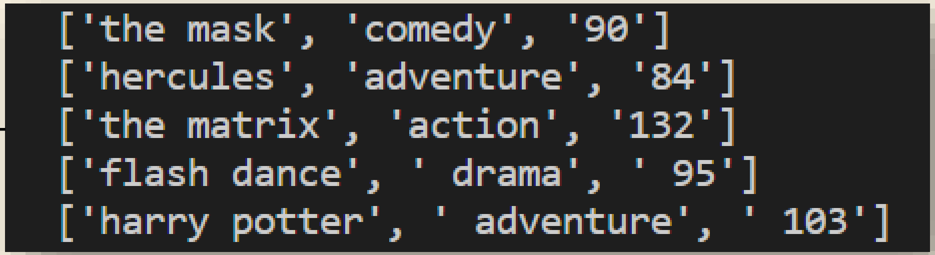
FILES


file_examples.py

```python
import csv

def main():
    with open('videos.csv') as video_file:
        csv_reader = csv.reader(video_file)

        next(csv_reader)

        for line in csv_reader:
            print(line)

main()
```

This line will make the reader skip the first line which holds the field names

This is the output

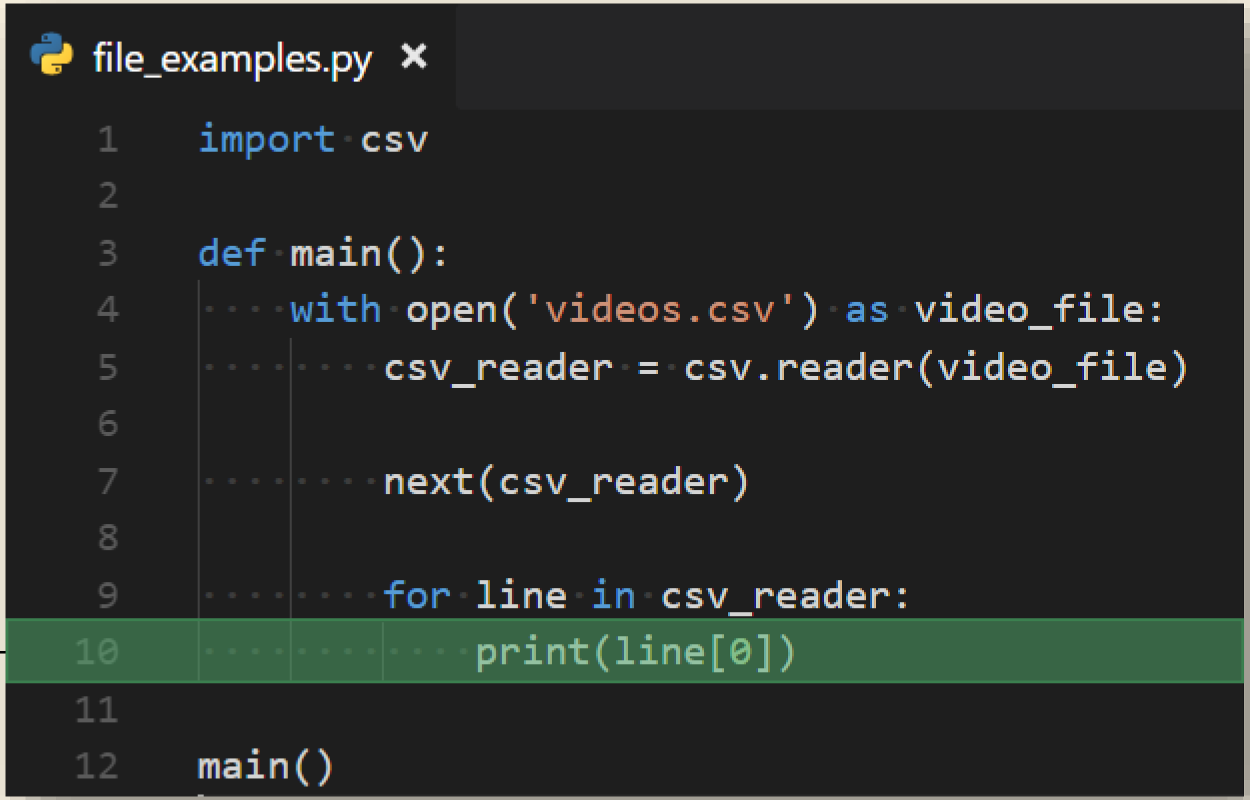


```
['the mask', 'comedy', '90']
['hercules', 'adventure', '84']
['the matrix', 'action', '132']
['flash dance', ' drama', ' 95']
['harry potter', ' adventure', ' 103']
```

FILES

- Since every line is represented as a list we can use indices in each line

This will only print the title of each video



file_examples.py

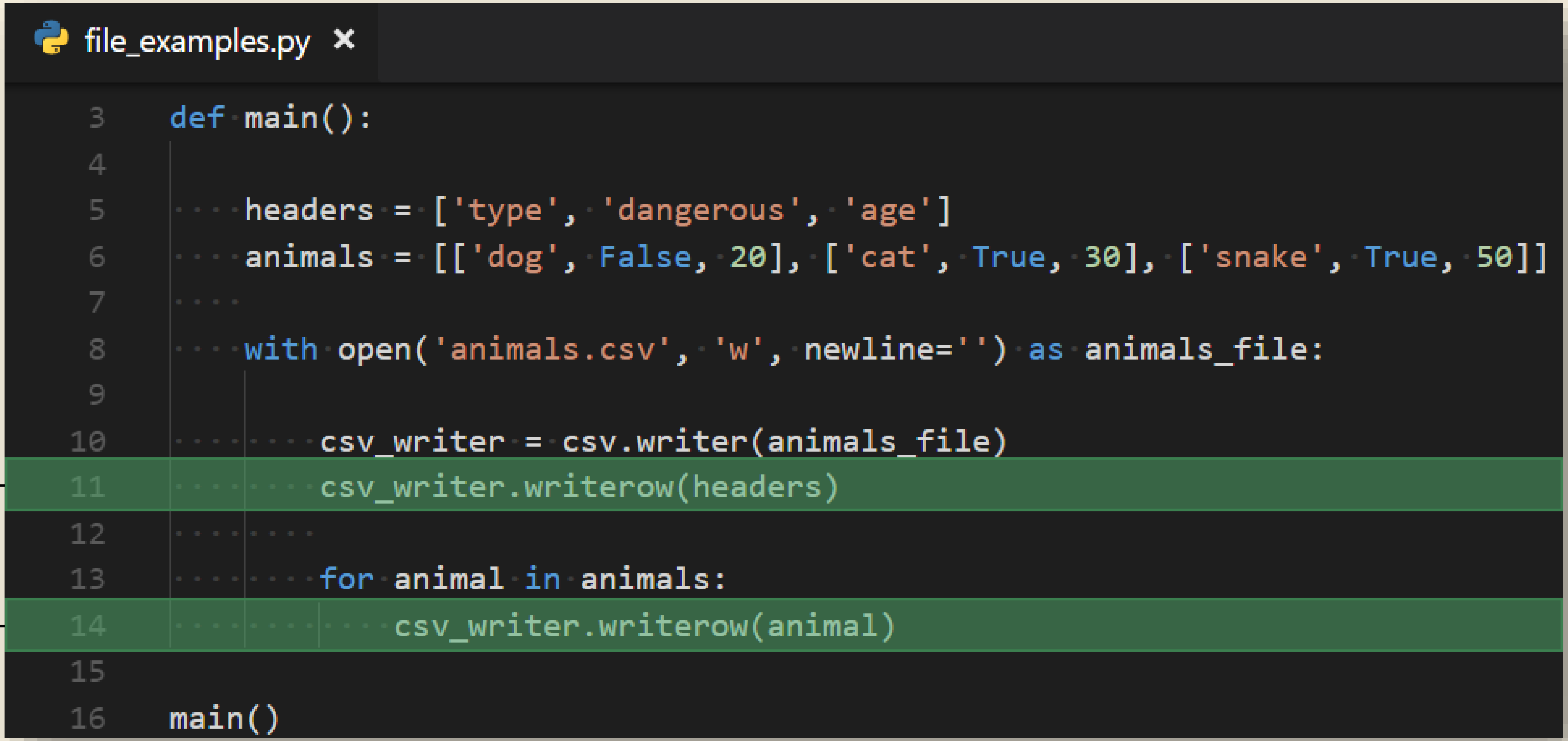
```python
import csv

def main():
    with open('videos.csv') as video_file:
        csv_reader = csv.reader(video_file)

        next(csv_reader)

        for line in csv_reader:
            print(line[0])

main()
```

FILES

- This example shows how we can write to a csv file from a list of lists

We start by adding the header fields to the file

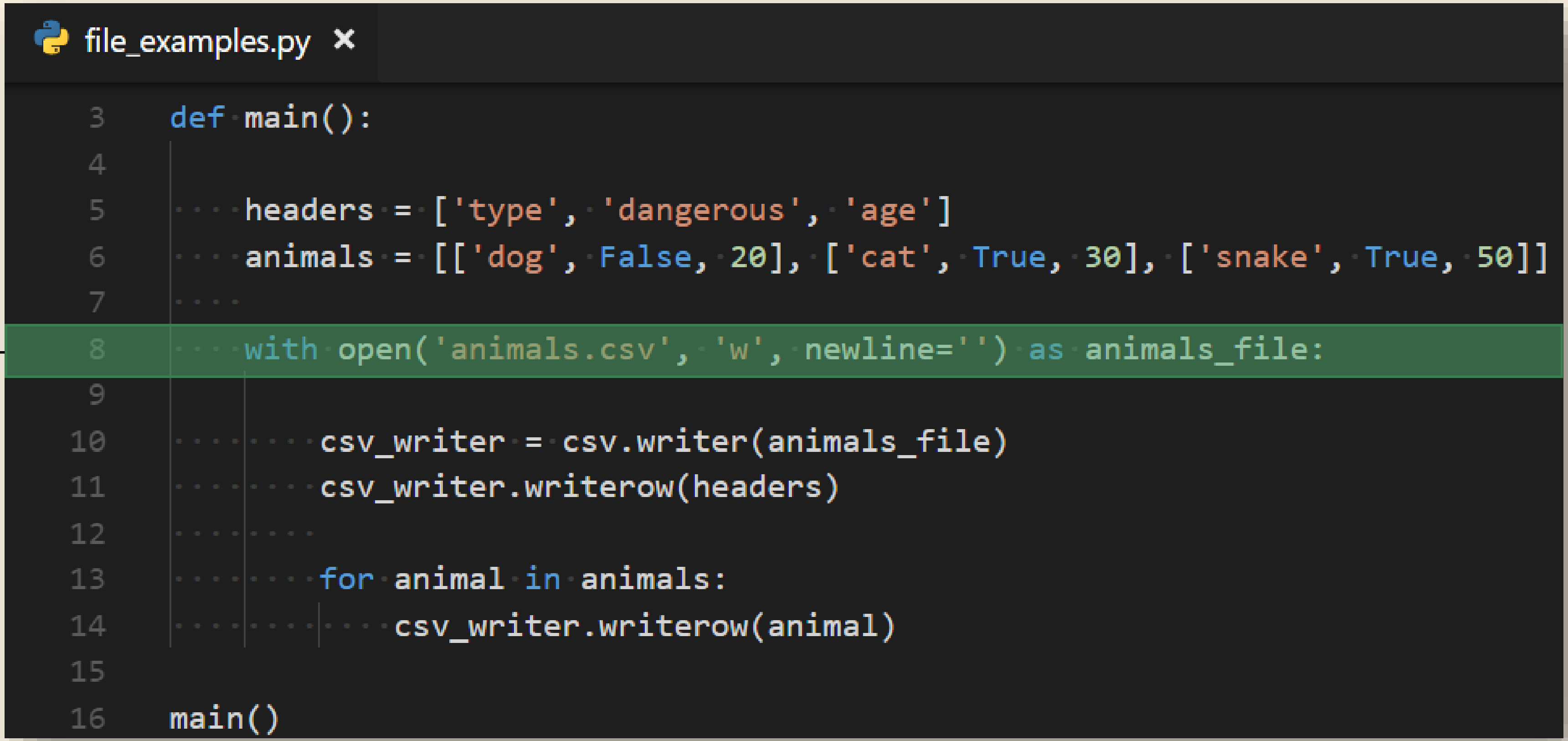
Here we write each animal to the file



```python
def main():

    headers = ['type', 'dangerous', 'age']
    animals = [['dog', False, 20], ['cat', True, 30], ['snake', True, 50]]

    with open('animals.csv', 'w', newline='') as animals_file:

        csv_writer = csv.writer(animals_file)
        csv_writer.writerow(headers)

        for animal in animals:
            csv_writer.writerow(animal)

main()
```

FILES

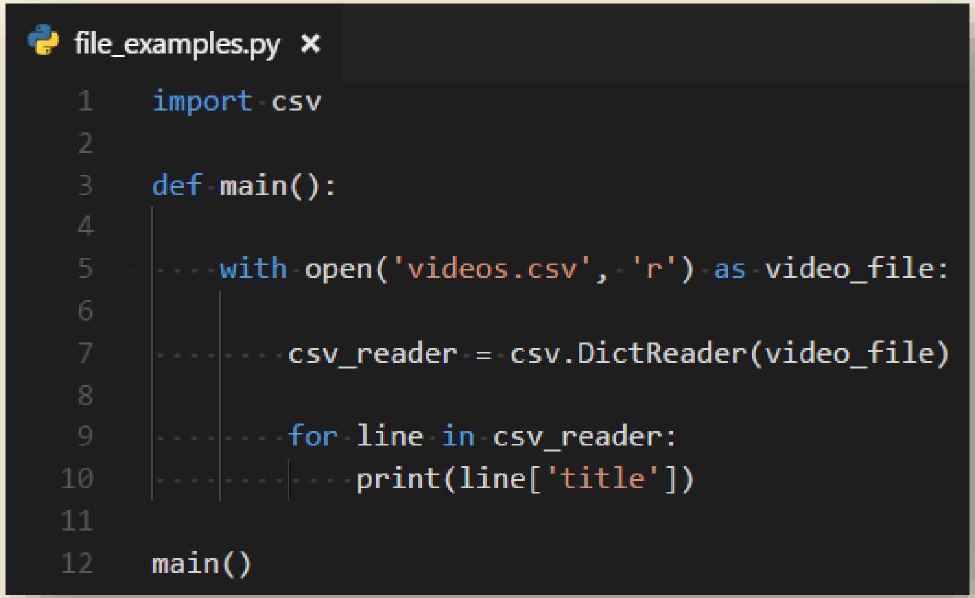
Note that the newline parameter is given because on **Windows** the csv writer adds a newline character to each line



```python
def main():

    headers = ['type', 'dangerous', 'age']
    animals = [['dog', False, 20], ['cat', True, 30], ['snake', True, 50]]
    
    with open('animals.csv', 'w', newline='') as animals_file:

        csv_writer = csv.writer(animals_file)
        csv_writer.writerow(headers)
        
        for animal in animals:
            csv_writer.writerow(animal)

main()
```

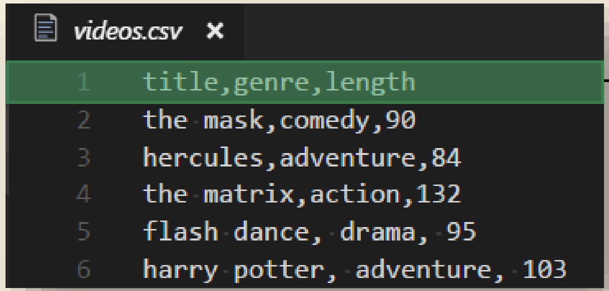
FILES

- Another way to read from csv files is to use the csv DictReader
  - It reades each line of the file as a dictionary and uses the field names contained in the first row of the csv file as the keys



file_examples.py

```python
import csv

def main():

    with open('videos.csv', 'r') as video_file:

        csv_reader = csv.DictReader(video_file)

        for line in csv_reader:
            print(line['title'])

main()
```



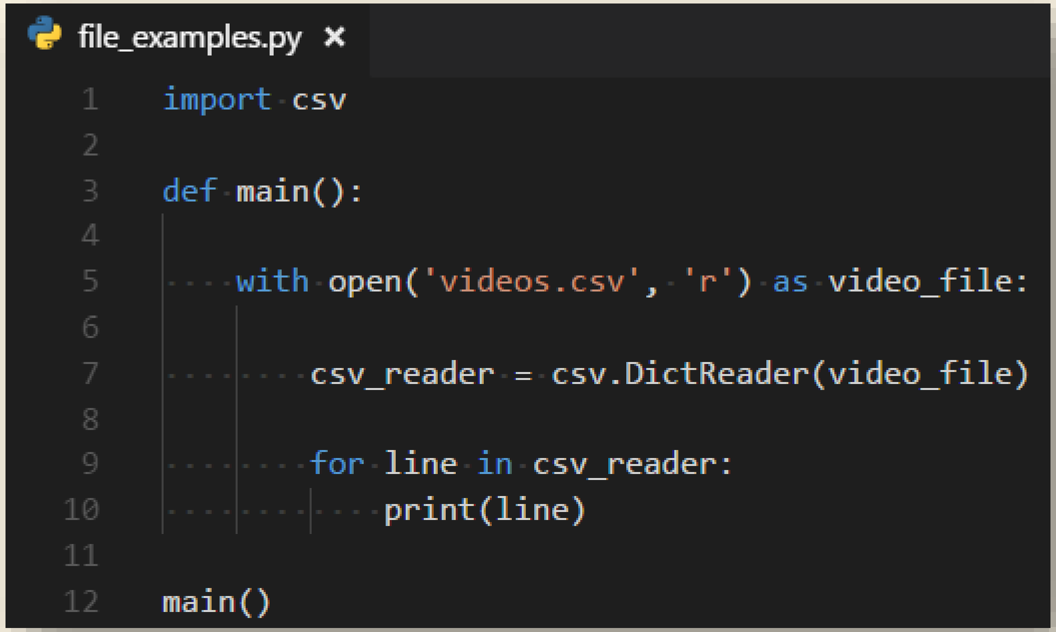
videos.csv

```
title,genre,length
the mask,comedy,90
hercules,adventure,84
the matrix,action,132
flash dance, drama, 95
harry potter, adventure, 103
```

The first line contains the field names. DictReader uses it as keys!

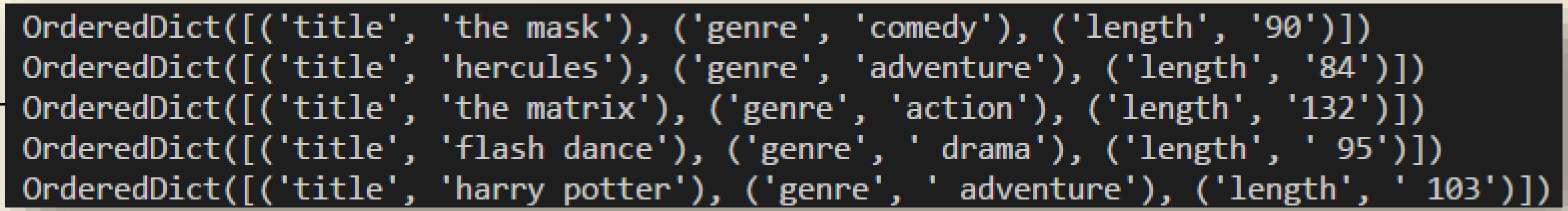
FILES

- Using the csv Dictreader can save us a lot of manual work but at first it may look a bit intimidating
  - It is usually a good idea to test modules such as the csv module with a small csv file/data set to get used to it



```python
import csv

def main():

    with open('videos.csv', 'r') as video_file:

        csv_reader = csv.DictReader(video_file)

        for line in csv_reader:
            print(line)

main()
```

This is the output if we print each line of the file using the DictReader



```
OrderedDict([('title', 'the mask'), ('genre', 'comedy'), ('length', '90')])
OrderedDict([('title', 'hercules'), ('genre', 'adventure'), ('length', '84')])
OrderedDict([('title', 'the matrix'), ('genre', 'action'), ('length', '132')])
OrderedDict([('title', 'flash dance'), ('genre', ' drama'), ('length', ' 95')])
OrderedDict([('title', 'harry potter'), ('genre', ' adventure'), ('length', ' 103')])
```