Lecture 1

Python review: Files, errors, data frames

Key Concepts

- Files
- Text files, Json files, python files, pickle files
- Saving text data
- Errors
- Data Frames

We need to learn how to

- Save/Load strings to text/pickle files
- Save/Load dictionaries to JSON files
- Save/Load data to a CSV file
- Handle errors in python scripts

Pre-requisites

- Familiarity with Python 3x
- Knowledge of while loops, for loops, data frames and file structure systems
- Understand what it means to import packages

What are computer files?

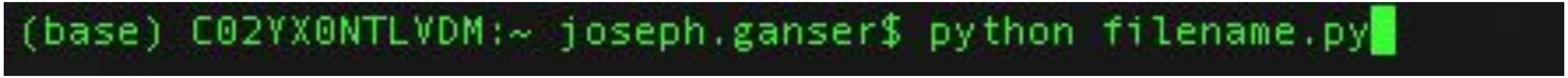
- Things like lists, tuples, dictionaries, sets, arrays offer only temporary data storage.

- When the computer is turned off, the RAM clears, and these objects are deleted.

- Txt, pickle, json, csv and other files offer long term storage to the hard drive.

Python Files on their own have the .py format; i.e. filename.py

- Code is written to a python file and saved in a way similar to a text file
- When a python program begins execution, it creates three file objects
  - Sys.stdin - input
  - Sys.stdout -output
  - Sys.stderr - error
- To run a python file in terminal we do; python filename.py



```
(base) C02YX0NTLVDM:~ joseph.ganser$ python filename.py
```

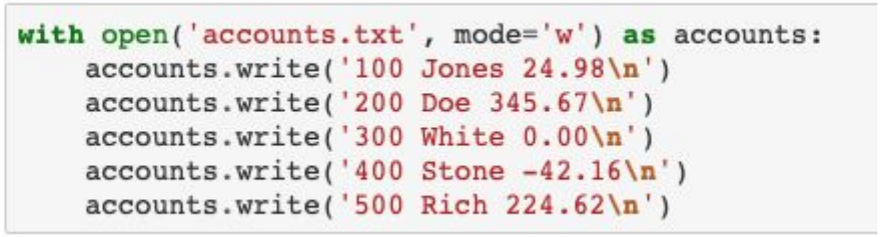
The Python ***with*** statement

- The with statement is used to acquire resources
  - Files, network connections, database connections, etc
- Acquires a resource and assign is corresponding object to a variable
- Calls the resource objects close method to release the resource

So what’s the code look like?

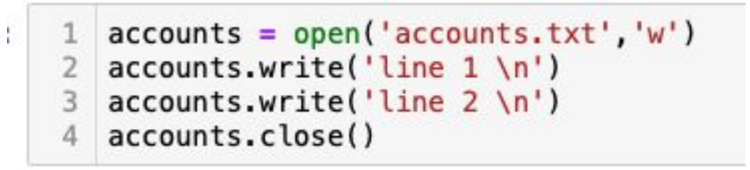
The Python ***with*** statement

- The with statement is used to acquire resources
    - Files, network connections, database connections, etc
- Acquires a resource and assign is corresponding object to a variable
- Calls the resource objects close method to release the resource
    - No need for .close() method



```
with open('accounts.txt', mode='w') as accounts:
    accounts.write('100 Jones 24.98\n')
    accounts.write('200 Doe 345.67\n')
    accounts.write('300 White 0.00\n')
    accounts.write('400 Stone -42.16\n')
    accounts.write('500 Rich 224.62\n')
```

Opening files without the ***with*** statement

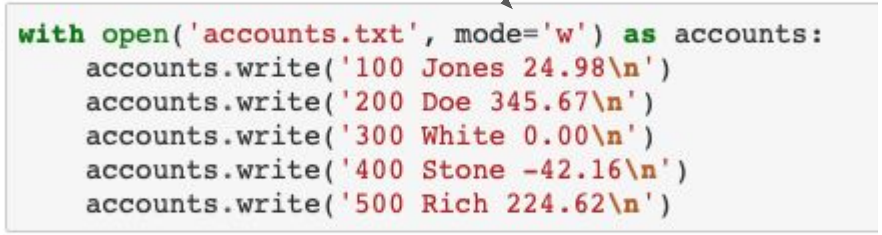


```
accounts = open('accounts.txt','w')
accounts.write('line 1 \n')
accounts.write('line 2 \n')
accounts.close()
```

- Need the .close() method to close file when done
  - Or it will stay open and can be messed up!

Using the python open function

- Notice there is a “mode” when we used the open function
- Writing had mode “w”
  - What about other modes?



```
with open('accounts.txt', mode='w') as accounts:
    accounts.write('100 Jones 24.98\n')
    accounts.write('200 Doe 345.67\n')
    accounts.write('300 White 0.00\n')
    accounts.write('400 Stone -42.16\n')
    accounts.write('500 Rich 224.62\n')
```

Using the python open function

- Different modes tell us different ways of using the open function

mode	Description
‘r’	Read file
‘w’	Write to file
‘a’	Append to end of file
‘r+’	Read & Write
‘w+’	Read & Write - pre-existing contents deleted
‘a+’	Append to end of file, create if it doesn’t exist already

File Types

- Txt - basic text file - used for holding unstructured data

- Json - (Javascript object notation) - stores data similar to a python dictionary
  - Useful for saving credentials, passwords, machine learning configurations, etc

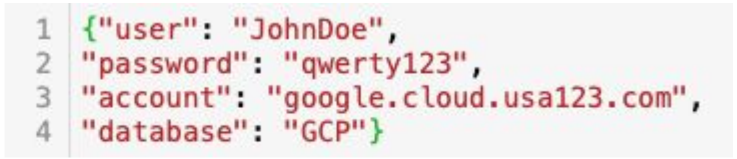
- Pickle - Data serialization technique that can store a wide variety of complex data structures, including machine learning models, data frames and others.
  - Can be hacked!

Json Files

JSON Files look just like python dictionaries

- Colon, comma, curly bracket structure
    - Data stored in key-value pairs

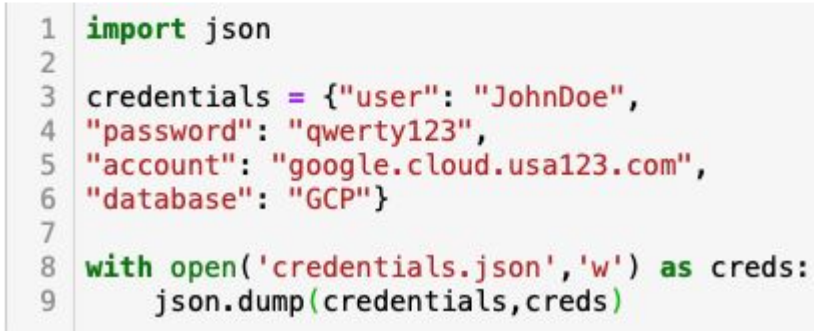
- Accessed just like python dictionaries



```json
{"user": "JohnDoe",
"password": "qwerty123",
"account": "google.cloud.usa123.com",
"database": "GCP"}
```

Json Files: Saving

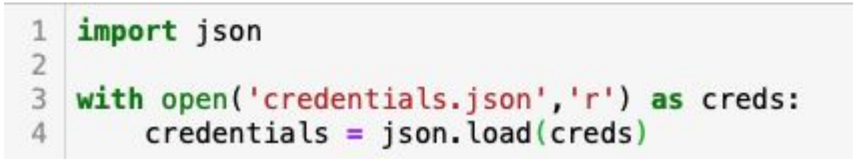
- Must import the json package
- Create an open object with ‘write’ method
  - Use json .dump method to save data



```python
import json

credentials = {"user": "JohnDoe",
"password": "qwerty123",
"account": "google.cloud.usa123.com",
"database": "GCP"}

with open('credentials.json','w') as creds:
    json.dump(credentials,creds)
```

Json Files: Loading

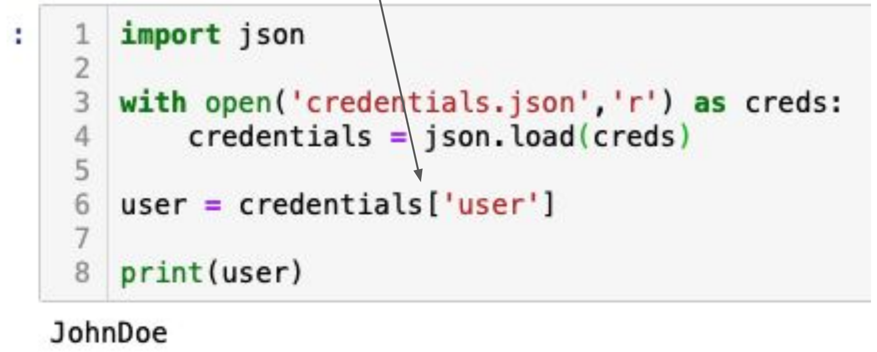
- Must import the json package
- Create an open object with ‘read’ method
  - Use json .load method to load data



```python
import json

with open('credentials.json','r') as creds:
    credentials = json.load(creds)
```

Json Files: Accessing

- Treat just like a python dictionary
  - Access key value pairs using square brackets



```python
import json

with open('credentials.json','r') as creds:
    credentials = json.load(creds)

user = credentials['user']

print(user)
```

```
JohnDoe
```

Pickle Files

- Useful and very old data serialization technique going back to the days of the ‘C’ language

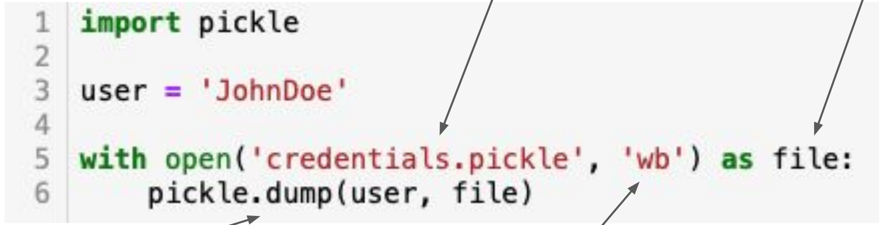
- Loads and saves data similar to how json files are loaded/saved

- Can save pretty much anything in a pickle format, though the best choice depends upon circumstances.
  - Good for saving machine learning models!
  - Typically save objects such as strings, dictionaries, sets, dataframes, booleans, numbers, etc

- Not good for using cross languages (e.g. making a python pickle file and using it in Java)

Pickle Files: Saving

- To save simply import pickle, specify file-name and object name

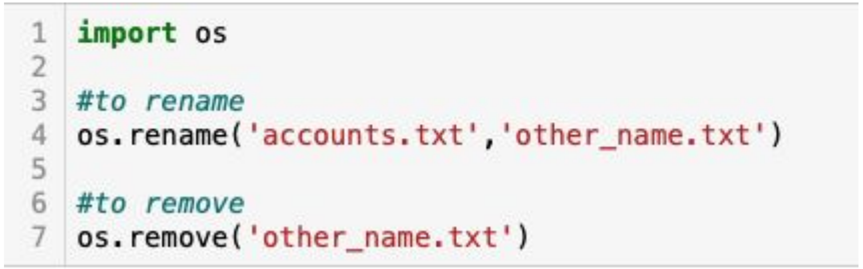


```python
import pickle

user = 'JohnDoe'

with open('credentials.pickle', 'wb') as file:
    pickle.dump(user, file)
```

- Use pickles .dump method to save
- Use ‘wb’ method to write binary - better for use across multiple operatings systems

Other file techniques: Remove and rename

- Must import ‘os’ package



```python
import os

#to rename
os.rename('accounts.txt','other_name.txt')

#to remove
os.remove('other_name.txt')
```

How do we deal with code errors in real time?

- Sometimes you can’t just re-write the script
- Instead you build code that handles specific types of errors for you

Examples of Errors

- FileNotFoundError - non existent file

- Permission error - no permissions to access specific data
  - E.g. An API receiving the wrong password

- ValueError - right type but inappropriate value
  - E.G. trying to pass a string to a number function

Error Handling: Clauses

There are 4 fundamental clauses used for handling errors

- Try
- Except
- Else
- Finally

Error Handling: Clauses

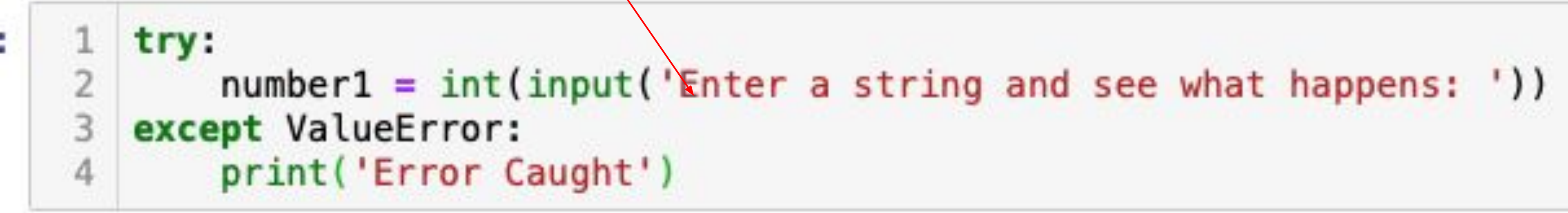
- Try: Makes the computer attempt a line of code. Typically followed by Except Clause

Error Handling: Clauses

- Except: comes After Try clause, handle in case of error
    - We can specify specific types of errors to handle

Error Handling: Clauses

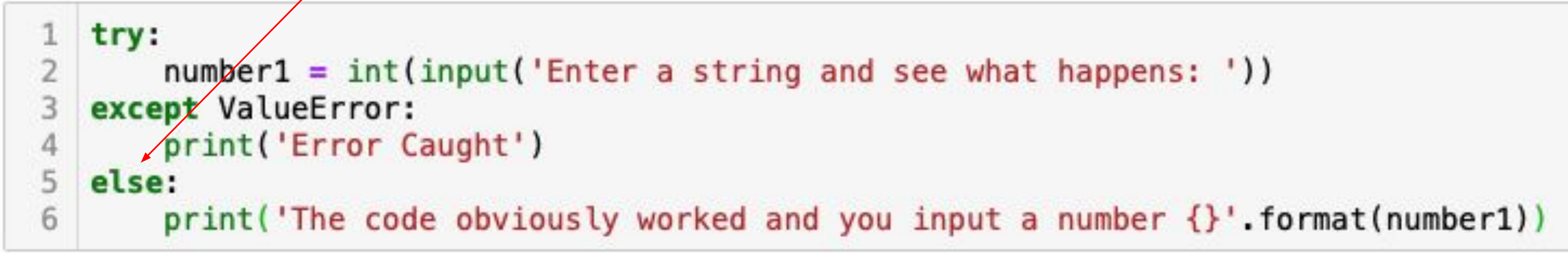
- Except: comes After Try clause, handle in case of error
  - We can specify specific types of errors to handle



```
try:
    number1 = int(input('Enter a string and see what happens: '))
except ValueError:
    print('Error Caught')
```

Error Handling: Clauses

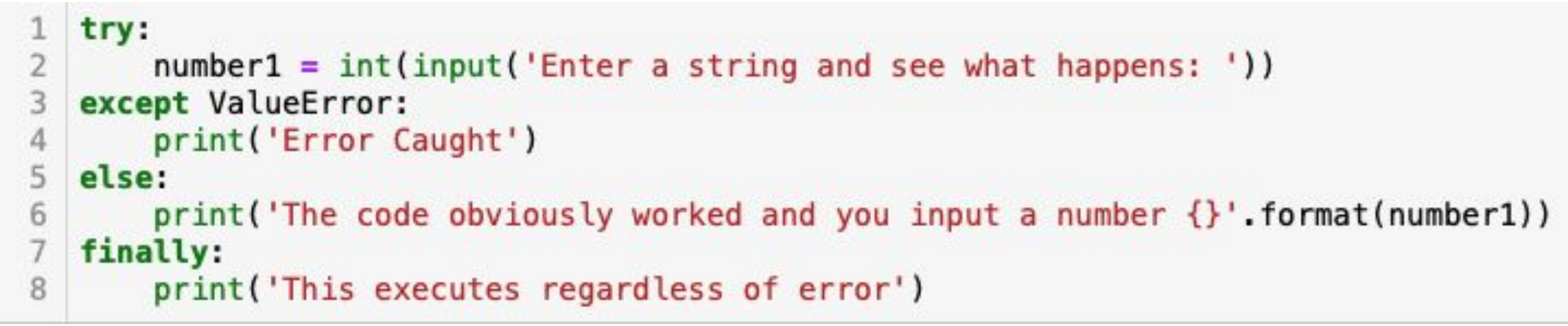
- Else: specifies code that should execute only if the code in the try suite did NOT raise exceptions



```python
try:
    number1 = int(input('Enter a string and see what happens: '))
except ValueError:
    print('Error Caught')
else:
    print('The code obviously worked and you input a number {}'.format(number1))
```

Error Handling: Clauses

- finally: specifies code that should execute only if the code in the try suite did NOT raise exceptions



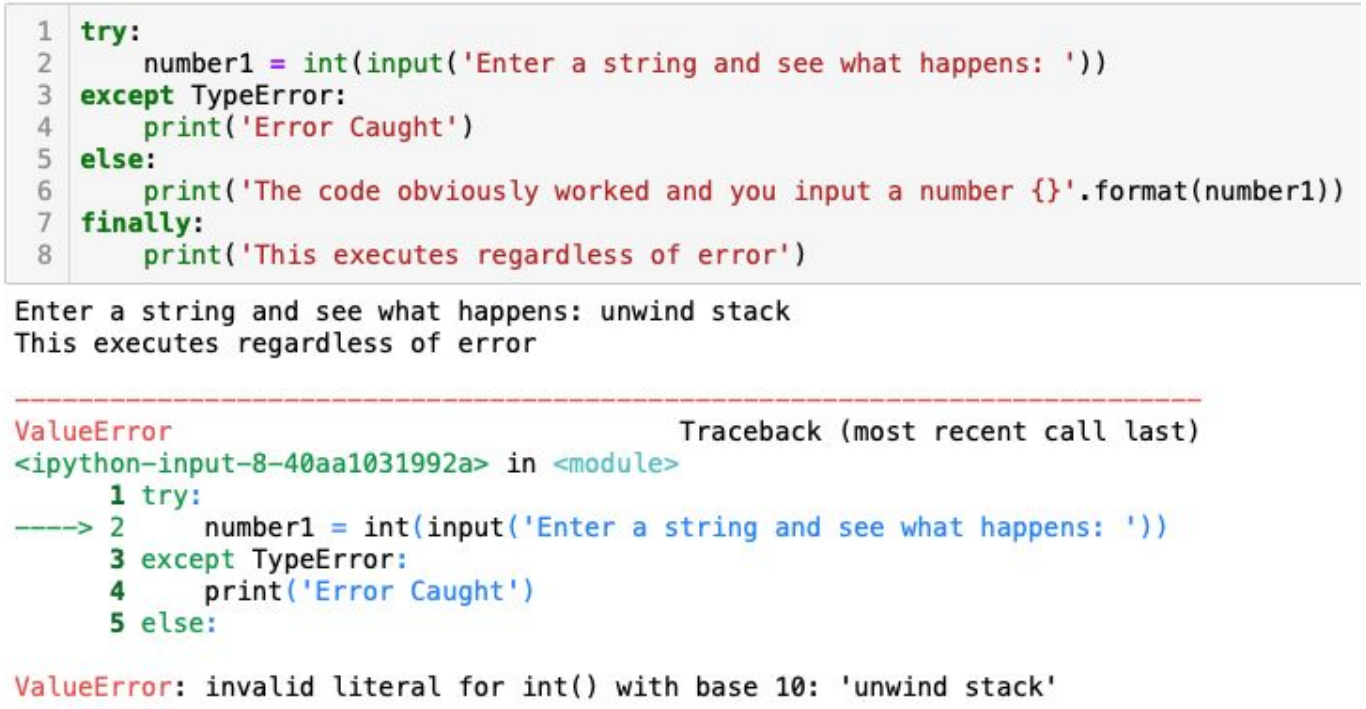
```python
try:
    number1 = int(input('Enter a string and see what happens: '))
except ValueError:
    print('Error Caught')
else:
    print('The code obviously worked and you input a number {}'.format(number1))
finally:
    print('This executes regardless of error')
```

Raising Exceptions & Stack Unwinding

- When an exception (error) isn’t caught, it causes “stack unwinding”
- This process allows us to figure out what went wrong in the code

Raising Exceptions & Stack Unwinding

- When an exception (error) isn't caught, it causes "stack unwinding"
- This process allows us to figure out what went wrong in the code



```python
try:
    number1 = int(input('Enter a string and see what happens: '))
except TypeError:
    print('Error Caught')
else:
    print('The code obviously worked and you input a number {}'.format(number1))
finally:
    print('This executes regardless of error')
```

```
Enter a string and see what happens: unwind stack
This executes regardless of error

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-40aa1031992a> in <module>
      1 try:
----> 2     number1 = int(input('Enter a string and see what happens: '))
      3 except TypeError:
      4     print('Error Caught')
      5 else:

ValueError: invalid literal for int() with base 10: 'unwind stack'
```

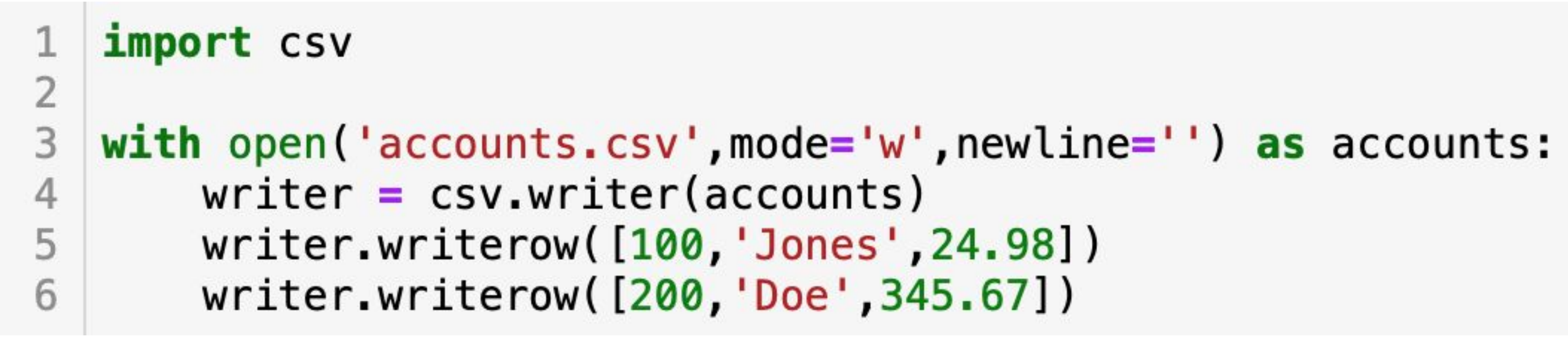
CSV Files

CSV Files: Comma separated Files

- Great for holding data frames
- Very similar to excel files
- Consists of rows, columns and data types

CSV Files

Let’s open a csv and write a few lines…



```python
import csv

with open('accounts.csv',mode='w',newline='') as accounts:
    writer = csv.writer(accounts)
    writer.writerow([100,'Jones',24.98])
    writer.writerow([200,'Doe',345.67])
```

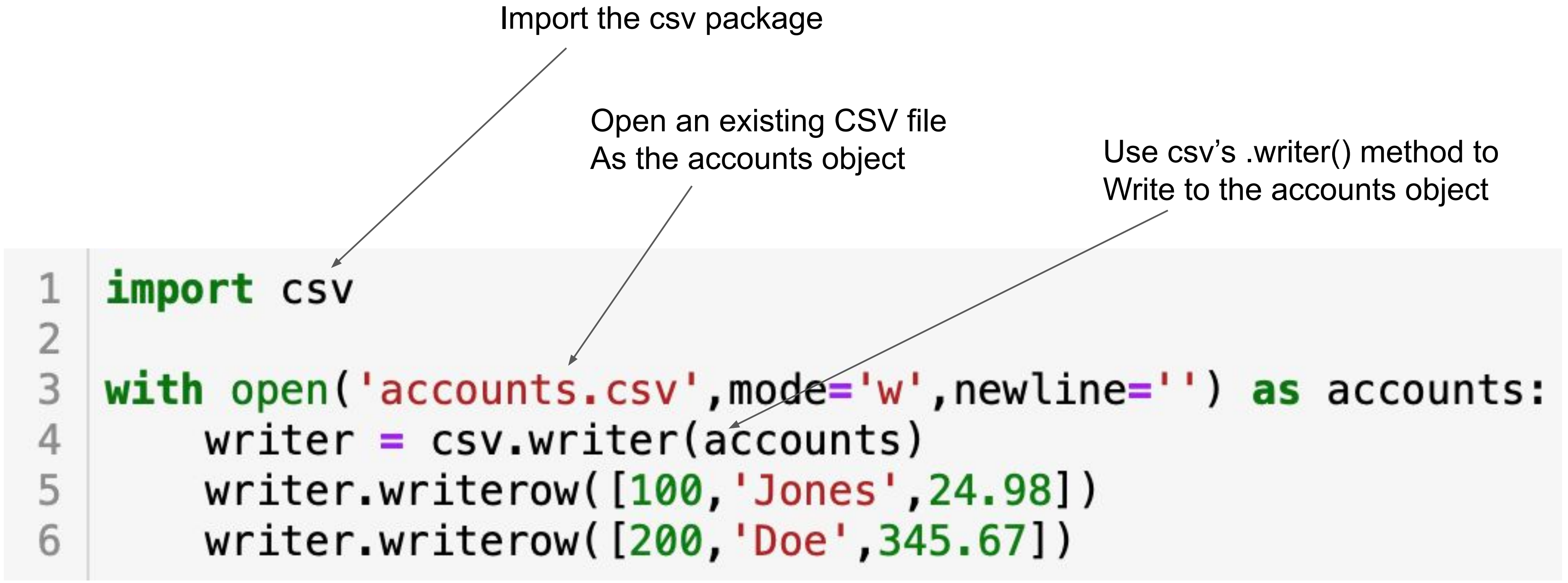
Import the csv package

```python
import csv

with open('accounts.csv',mode='w',newline='') as accounts:
    writer = csv.writer(accounts)
    writer.writerow([100,'Jones',24.98])
    writer.writerow([200,'Doe',345.67])
```

Import the csv package

Open an existing CSV file
As the accounts object

```python
import csv

with open('accounts.csv',mode='w',newline='') as accounts:
    writer = csv.writer(accounts)
    writer.writerow([100,'Jones',24.98])
    writer.writerow([200,'Doe',345.67])
```


Import the csv package

Open an existing CSV file
As the accounts object

Use csv’s .writer() method to
Write to the accounts object

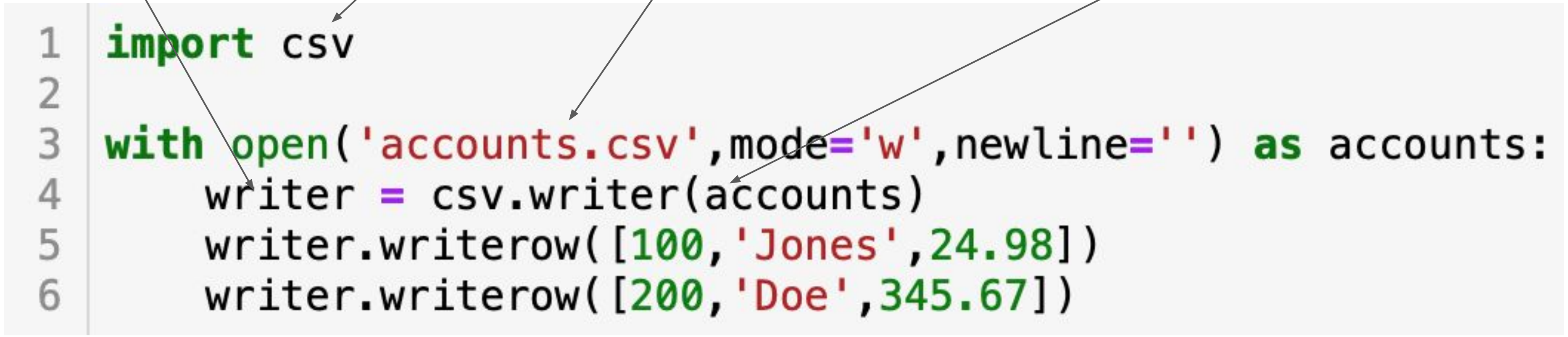
```
import csv

with open('accounts.csv',mode='w',newline='') as accounts:
    writer = csv.writer(accounts)
    writer.writerow([100,'Jones',24.98])
    writer.writerow([200,'Doe',345.67])
```

Import the csv package

Open an existing CSV file
As the accounts object

Instantiate the writer object

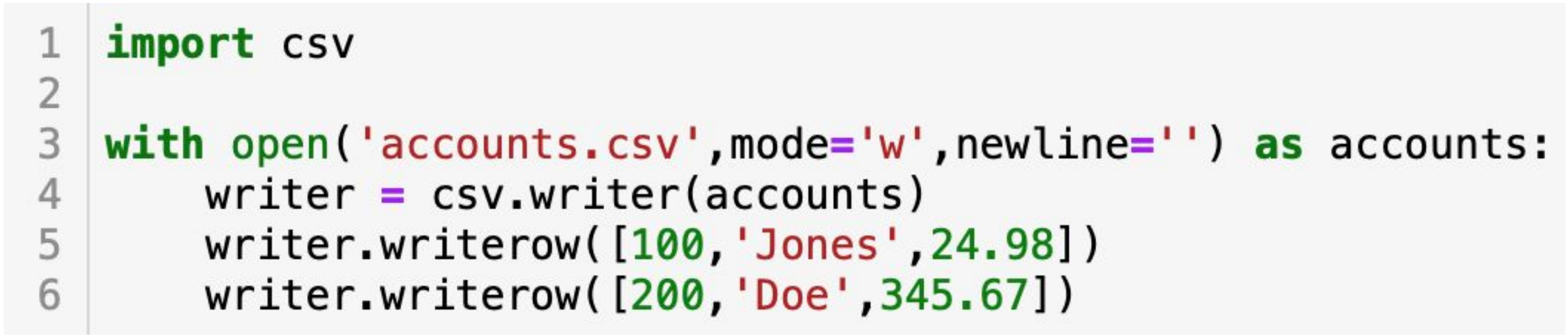
Use csv's .writer() method to
Write to the accounts object



```python
import csv

with open('accounts.csv',mode='w',newline='') as accounts:
    writer = csv.writer(accounts)
    writer.writerow([100,'Jones',24.98])
    writer.writerow([200,'Doe',345.67])
```

when writing to a data frame:

- All rows must have the same number of entries (e.g. 3 columns of entries)
    - If an entry is missing, put a null value
- Rows should have matching data types



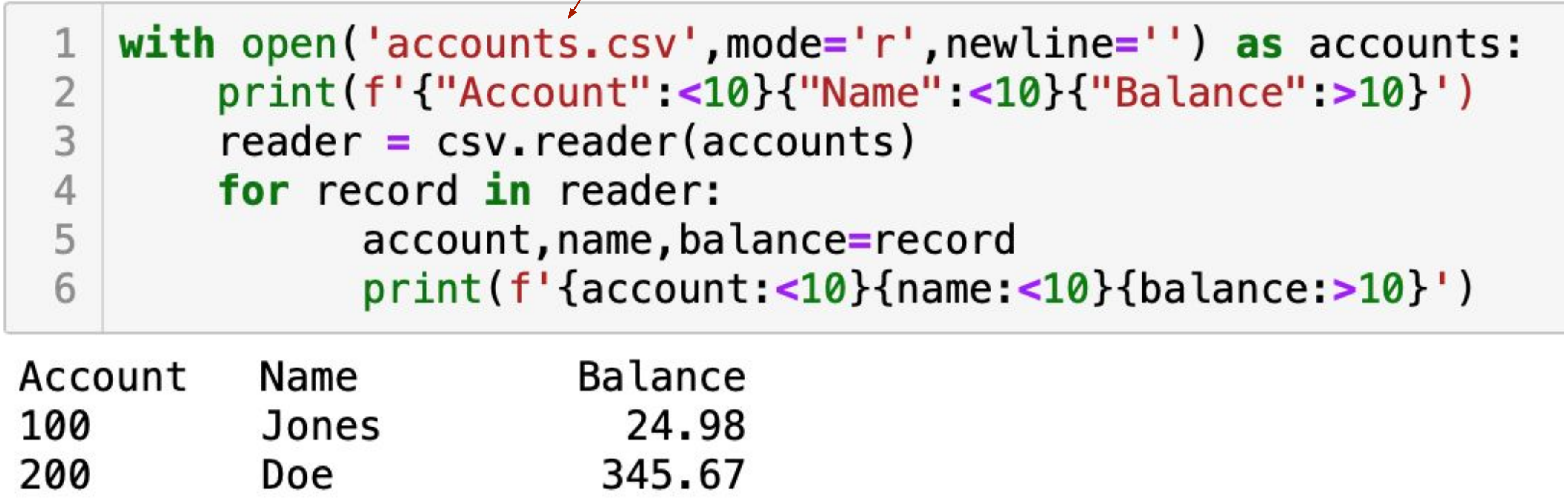
```python
import csv

with open('accounts.csv',mode='w',newline='') as accounts:
    writer = csv.writer(accounts)
    writer.writerow([100,'Jones',24.98])
    writer.writerow([200,'Doe',345.67])
```

Reading from a CSV

```python
with open('accounts.csv',mode='r',newline='') as accounts:
    print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
    reader = csv.reader(accounts)
    for record in reader:
        account,name,balance=record
        print(f'{account:<10}{name:<10}{balance:>10}')
```

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
```

Open csv file, mode ‘r’ to read, and newline=’’ to ensure newlines are processed properly

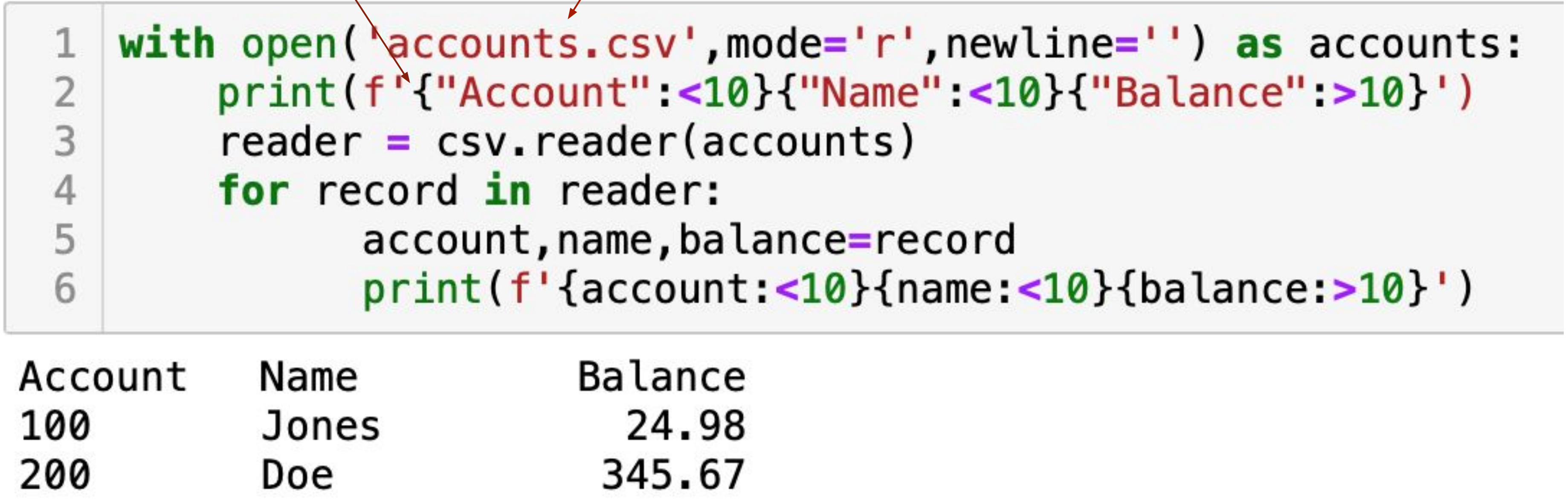


```python
with open('accounts.csv',mode='r',newline='') as accounts:
    print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
    reader = csv.reader(accounts)
    for record in reader:
          account,name,balance=record
          print(f'{account:<10}{name:<10}{balance:>10}')
```

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
```

Open csv file, mode ‘r’ to read, and newline=’’ to ensure newlines are processed properly

Print column names



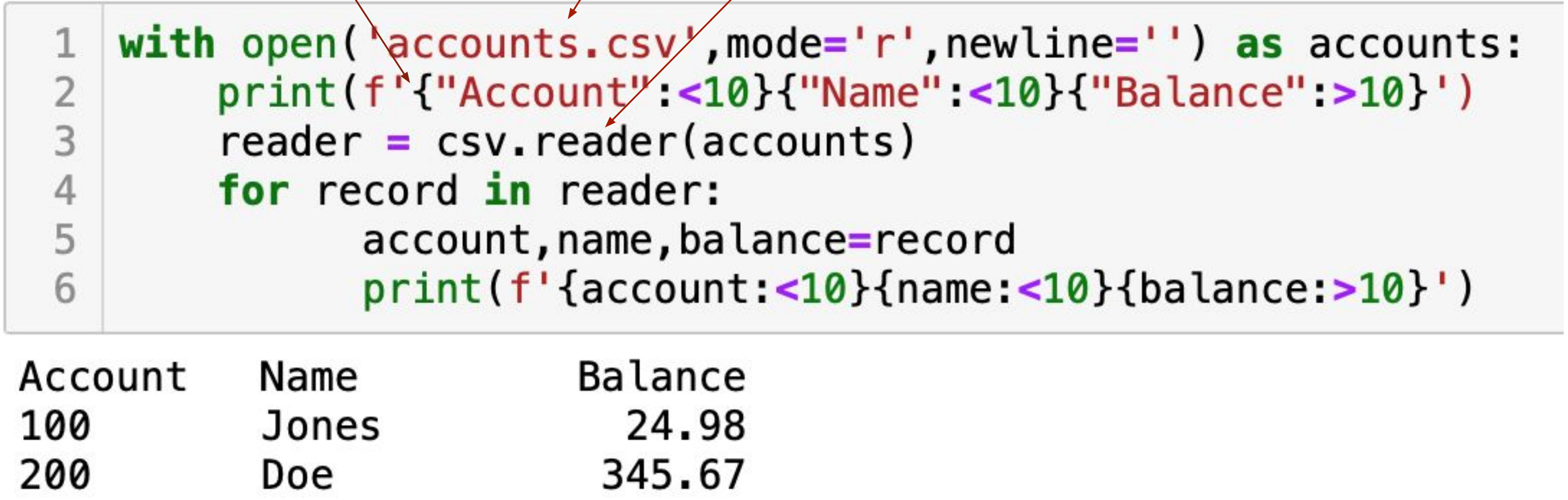
```python
with open('accounts.csv',mode='r',newline='') as accounts:
    print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
    reader = csv.reader(accounts)
    for record in reader:
        account,name,balance=record
        print(f'{account:<10}{name:<10}{balance:>10}')
```

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
```

Open csv file, mode ‘r’ to read, and newline=’’ to ensure newlines are processed properly

Print column names

Instantiate the reader object



```python
with open('accounts.csv',mode='r',newline='') as accounts:
    print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
    reader = csv.reader(accounts)
    for record in reader:
        account,name,balance=record
        print(f'{account:<10}{name:<10}{balance:>10}')
```

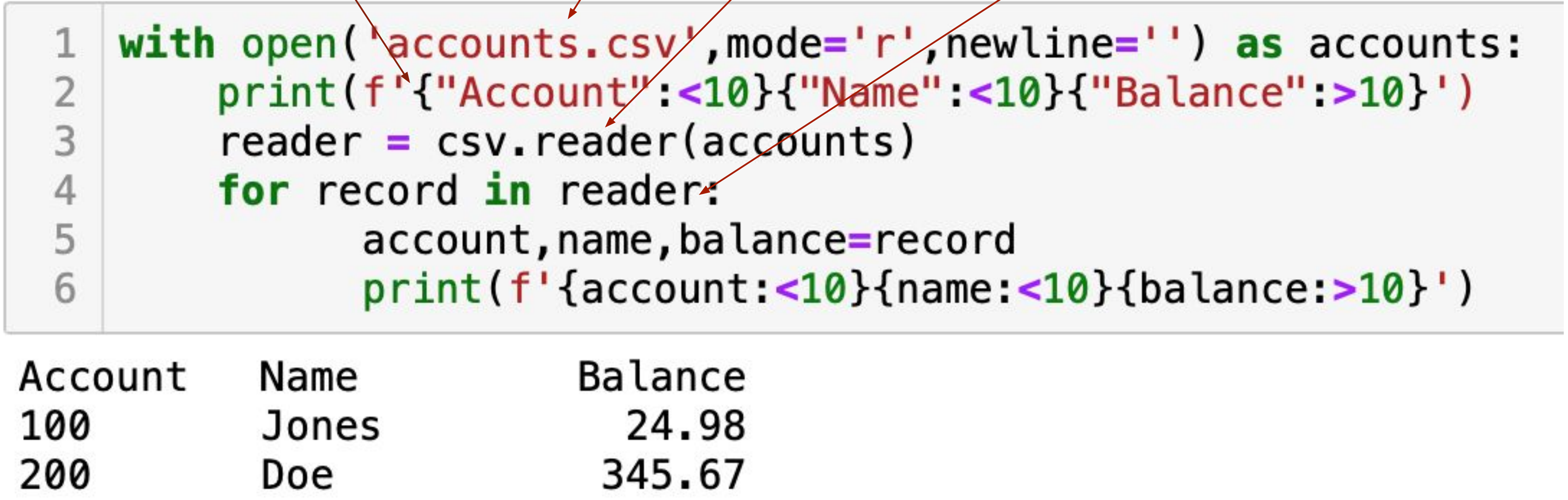
```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
```

Open csv file, mode ‘r’ to read, and newline=’’ to ensure newlines are processed properly

Print column names

Instantiate the reader object

Loop through each row to print values



```python
with open('accounts.csv',mode='r',newline='') as accounts:
    print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
    reader = csv.reader(accounts)
    for record in reader:
        account,name,balance=record
        print(f'{account:<10}{name:<10}{balance:>10}')
```

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
```

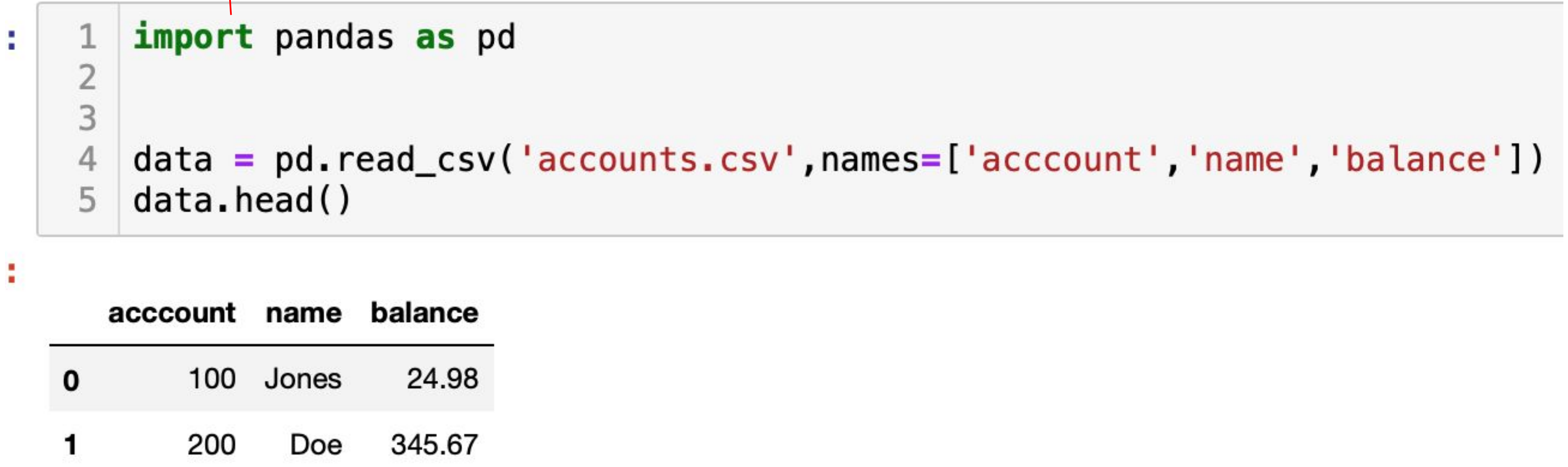
CSV Files & Pandas

Pandas

- Fundamental package in python used for data science

- Lots of built in functions, very useful to evaluate and transform dataframes

Import pandas package, give it an abbreviation

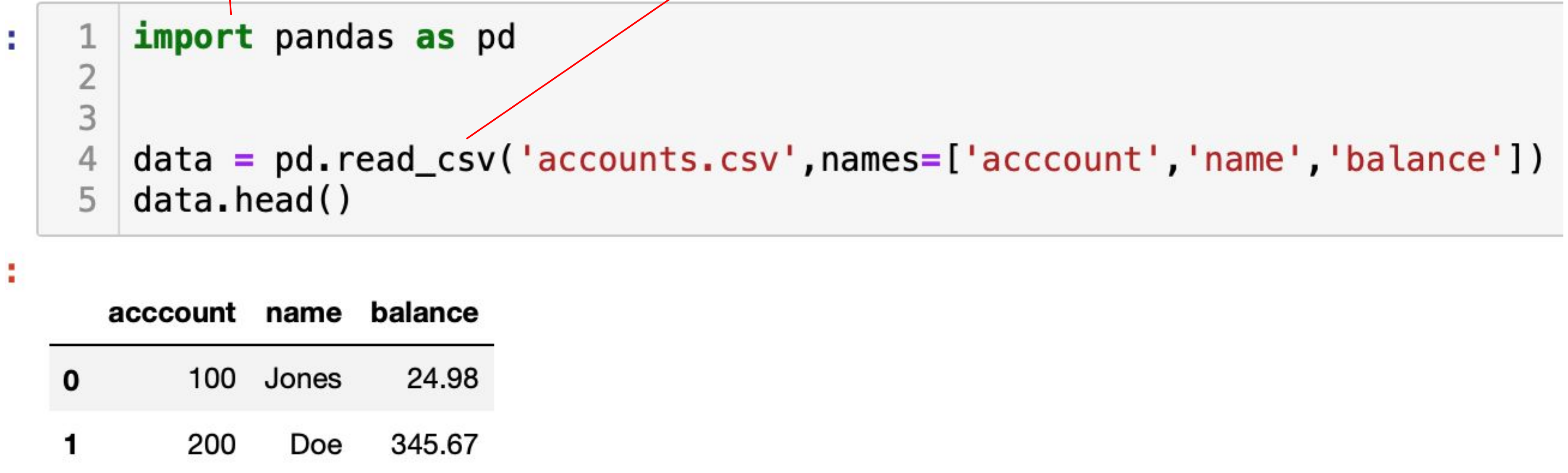


```
import pandas as pd


data = pd.read_csv('accounts.csv',names=['acccount','name','balance'])
data.head()
```

```
   acccount   name  balance
0       100  Jones    24.98
1       200    Doe   345.67
```

Import pandas package, give it an abbreviation

Use the read_csv function to read the CSV file



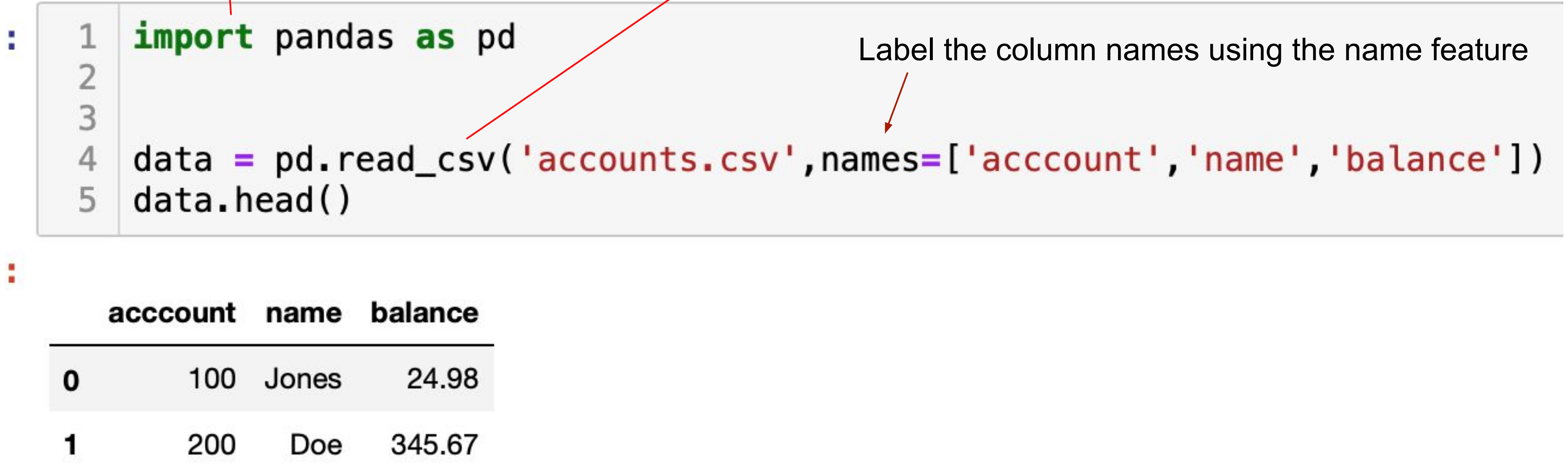
```
import pandas as pd


data = pd.read_csv('accounts.csv',names=['acccount','name','balance'])
data.head()
```

	acccount	name	balance
0	100	Jones	24.98
1	200	Doe	345.67

Import pandas package, give it an abbreviation

Use the read_csv function to read the CSV file

Label the column names using the name feature



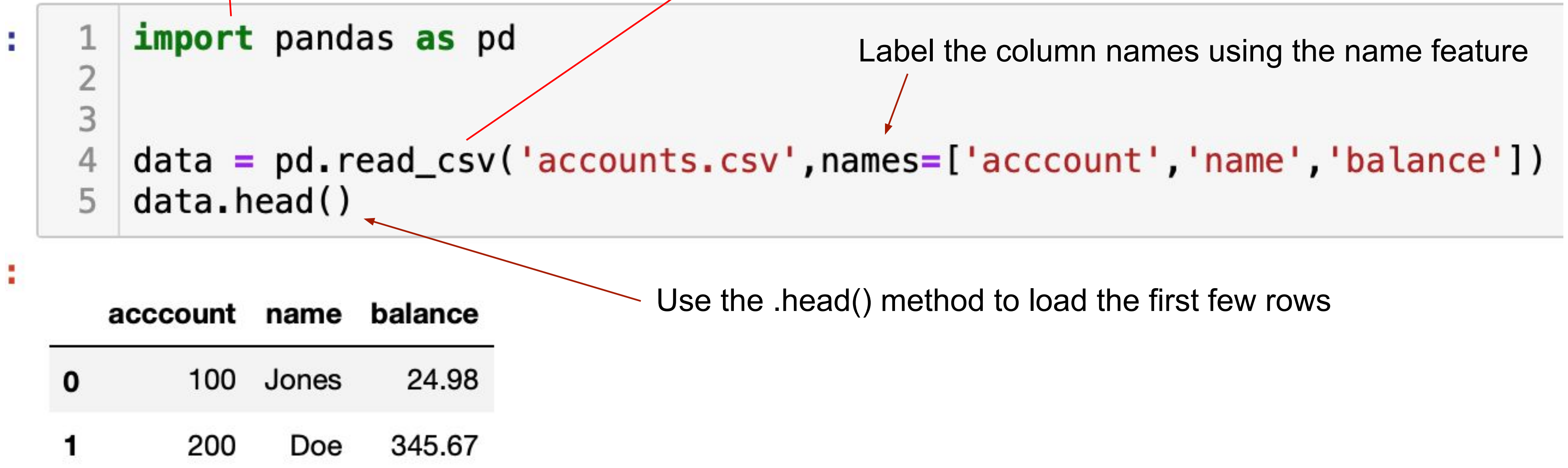
```
import pandas as pd


data = pd.read_csv('accounts.csv',names=['acccount','name','balance'])
data.head()
```

```
   acccount   name  balance
0       100  Jones    24.98
1       200    Doe   345.67
```

Import pandas package, give it an abbreviation

Use the read_csv function to read the CSV file

Label the column names using the name feature



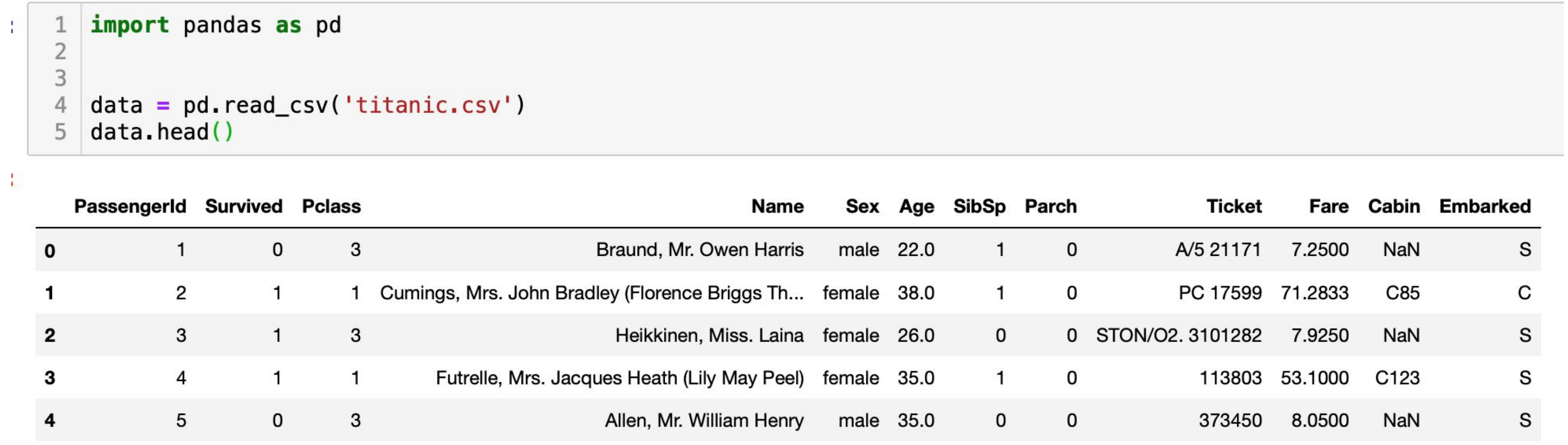
```python
import pandas as pd


data = pd.read_csv('accounts.csv',names=['acccount','name','balance'])
data.head()
```

```
   acccount   name  balance
0       100  Jones    24.98
1       200    Doe   345.67
```

Use the .head() method to load the first few rows

CSV Files & Pandas

Example link:
https://github.com/awesomedata/awesome-public-datasets/blob/master/Datasets/titanic.csv.zip

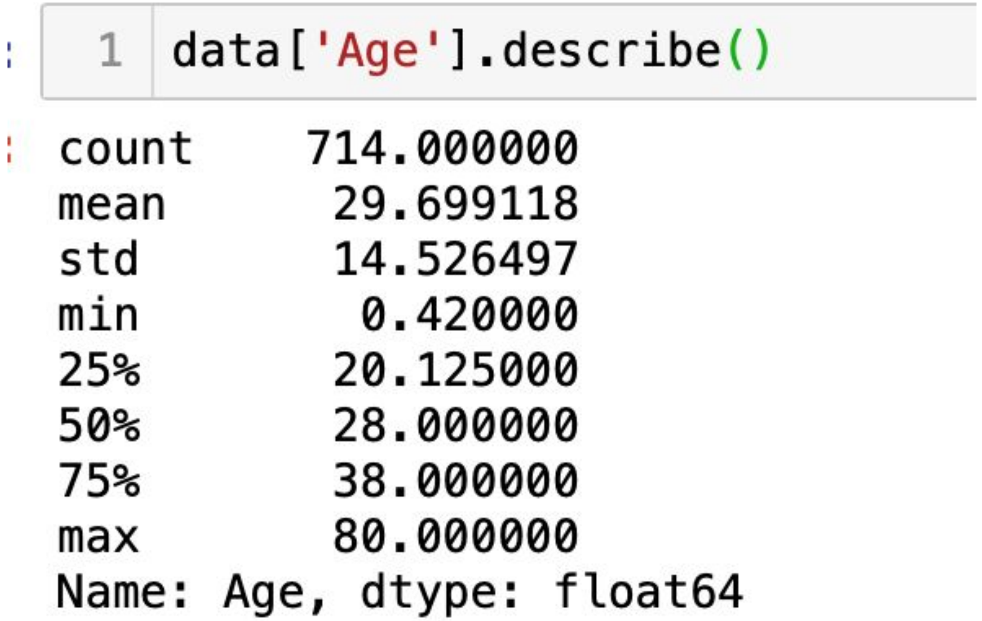


```python
import pandas as pd


data = pd.read_csv('titanic.csv')
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

CSV Files & Pandas

We can use pandas functions like .describe() on a specified column (e.g. Age) to see properties of the data

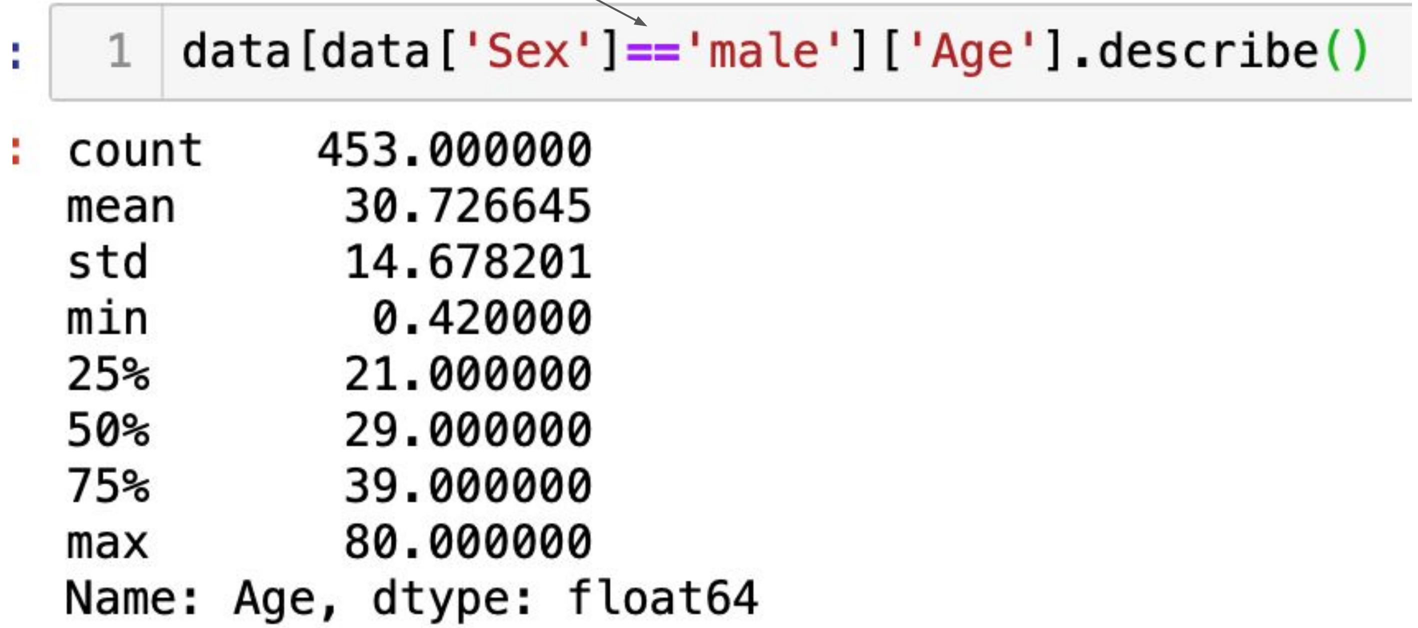


```
data['Age'].describe()
```

```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

CSV Files & Pandas

We can specify rows for which a column has a specific type using boolean logic

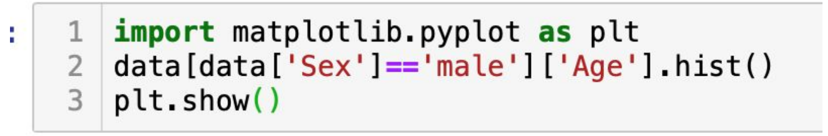


```
data[data['Sex']=='male']['Age'].describe()
```

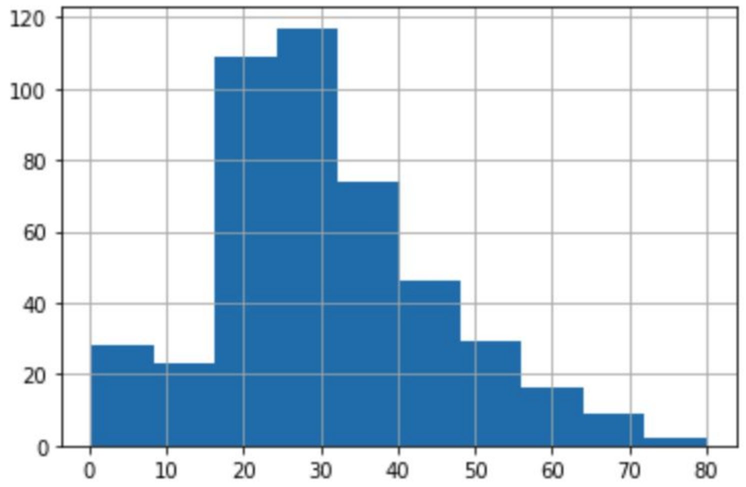
```
count    453.000000
mean      30.726645
std       14.678201
min        0.420000
25%       21.000000
50%       29.000000
75%       39.000000
max       80.000000
Name: Age, dtype: float64
```

CSV Files & Pandas

We can even get histograms of data within the data frame using the .hist() function (must also import matplotlib)



```
import matplotlib.pyplot as plt
data[data['Sex']=='male']['Age'].hist()
plt.show()
```



Source Info

- Textbook:
  Intro to Python for Computer Science and Data Science
  Ch 9.8 - 9.12
  By Paul & Harvey Deitel
  2020
  ISBN10: 0-13-540467-3

- Website: https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html