Importing entire text files

Here you'll get experience opening a text file, printing its contents to the shell and, finally, closing it.

In [6]:
```
filename = 'chicago_crime.txt'
file = open(filename, mode='r')
text = file.read()
file.close()
```

In [110]:
```
print(text[:19])
# burada n'nci elemanin ciktini al dersek sadece bir karakter cikartir, c
```

```
Time    Percent
99      0.0
```

In [8]:
```
print(file.closed)
```

```
True
```

In [12]:
```
with open('chicago_crime.txt') as file:
    print(file.readline())

    #print(file.read())
```

```
Date,Block,Primary Type,Description,Location Description,Arrest,Domest
ic,District
```

Using NumPy to import flat files

In [15]:
```
import numpy as np
```

In [19]:
```
filename = 'MINST.txt'
data = np.loadtxt(filename, delimiter=',')
print(data)
```

In [21]:
```
print(type(data))
```

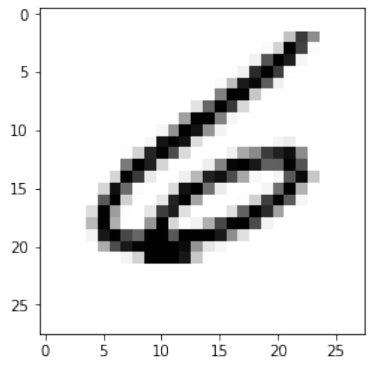
```
<class 'numpy.ndarray'>
```

In [29]:
```
im = data[21, 1:]
im_sq = np.reshape(im, (28, 28))
```

In [30]:
```
import matplotlib.pyplot as plt
```

In [31]:
```
plt.imshow(im_sq, cmap='Greys', interpolation='nearest')
```

Out[31]:
```
<matplotlib.image.AxesImage at 0x106db6b00>
```



```
# Customizing your NumPy import
```

# Customizing your NumPy import

```
In [38]: data = np.loadtxt(filename, delimiter=',', skiprows=1, usecols=[0, 2])
```

```
In [54]: print(data[3])

['99' '0.067']
```

```
In [48]: data = np.loadtxt('seaslugs.txt', delimiter='\t', dtype=str)
```

```
In [53]: print(data[0])

['Time' 'Percent']
```

```
In [62]: data_float = np.loadtxt('seaslugs.txt', delimiter='\t', dtype=float, skip
```

```
In [65]: print(data_float[10])

[0.    0.533]
```

# Working with mixed datatypes (1)

```
In [ ]: data = np.genfromtxt('titanic.csv', delimiter=',', names=True, dtype=None
```

```
In [200]: np.shape(data)

Out[200]: (5, 11)
```

```
In [ ]: print(data['Survived'])
```

```
In [74]: print(data[7])

(8, 0, 3, b'male', 2., 3, 1, b'349909', 21.075, b'', b'S')
```

# Working with mixed datatypes (2)

```
In [ ]: d = np.recfromcsv('titanic.csv')
```

```
In [83]: print(d[:3])

[(1, 0, 3, b'male', 22., 1, 0, b'A/5 21171',  7.25  , b'', b'S')
 (2, 1, 1, b'female', 38., 1, 0, b'PC 17599', 71.2833, b'C85', b'C')
 (3, 1, 3, b'female', 26., 0, 0, b'STON/O2. 3101282',  7.925 , b'', b'
S')]
```

# Using pandas to import flat files as DataFrames (1)

```
In [84]: import pandas as pd
```

```
In [85]: titanic = pd.read_csv('titanic.csv')
```

```
In [86]: titanic.head()
```

Out[86]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarl
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	male	35.0	0	0	373450	8.0500	NaN	

Type *Markdown* and LaTeX: α^2

```
In [ ]: # Dealing with missing values and incorrect data types /// baska bir Data
        # df = pd.read_csv("data/cereal.csv", skiprows = 1, na_values = ['no infc
```

# Using pandas to import flat files as DataFrames (2)

```
In [87]: file = 'titanic.csv'
```

```
In [88]: data = pd.read_csv(file, nrows=5, header=None)
```

```
In [94]: data_array = data.values
```

```
In [93]: print(type(data_array))

         <class 'numpy.ndarray'>
```

# Customizing your pandas import

```
In [ ]: file = 'titanic_corrupt.txt'
        data = pd.read_csv(file, sep='\t', comment='#', na_values='Nothing')
        print(data.head())
        pd.DataFrame.hist(data[['Age']])
        plt.xlabel('Age (years)')
        plt.ylabel('count')
        plt.show()
```

Introduction to other file types

# Not so flat any more

In Chapter 1, you learned how to use the IPython magic command ! ls to explore your current working directory. You can also do this natively in Python using the library os, which consists of miscellaneous operating system interfaces.

The first line of the following code imports the library os, the second line stores the name of the current directory in a string called wd and the third outputs the contents of the directory in a list to the shell.

```
import os
wd = os.getcwd()
os.listdir(wd)
```

```
In [120]: pwd

Out[120]: '/Users/onlyone/Documents'
```

```
In [115]: import os
```

```
In [116]: wd = os.getcwd()
          os.listdir(wd)
```

# Loading a pickled file

There are a number of datatypes that cannot be saved easily to flat files, such as lists and dictionaries. If you want your files to be human readable, you may want to save them as text files in a clever manner. JSONs, which you will see in a later chapter, are appropriate for Python dictionaries.

However, if you merely want to be able to import them into Python, you can serialize them. All this means is converting the object into a sequence of bytes, or a bytestream.

```
In [ ]: import pickle

        with open('data.pkl', 'rb') as file:
            d = pickle.load(file)

        print(d)
        print(type(d))
```

```
In [ ]: {'June': '69.4', 'Aug': '85', 'Airline': '8', 'Mar': '84.4'}
        <class 'dict'>
```

Listing sheets in Excel files

```
In [121]: import pandas as pd
```

```
In [145]: file='battledeath.xlsx'
```

```
In [146]: hakan=pd.ExcelFile(file)
```

```
In [140]: print(hakan.sheet_names)

['2002', '2004']
```

```
In [143]: print(hakan)

<pandas.io.excel.ExcelFile object at 0x115b59cf8>
```

Importing sheets from Excel files

```
In [154]: hakan1=hakan.parse('2002') # sheet name as string # parse = ayristirmak
          print(hakan1.head())

  War, age-adjusted mortality due to       2002
0                        Afghanistan  36.083990
1                            Albania   0.128908
2                            Algeria  18.314120
3                            Andorra   0.000000
4                             Angola  18.964560
```

```
In [155]: hakan2=hakan.parse(0) # sheet name as a float
          print(hakan2.head())

  War, age-adjusted mortality due to       2002
0                        Afghanistan  36.083990
1                            Albania   0.128908
2                            Algeria  18.314120
3                            Andorra   0.000000
4                             Angola  18.964560
```

Customizing your spreadsheet import

```
In [162]: hakan1 = hakan.parse(0, skiprows=1, names=['Country', 'AMM due to War (20
          print(hakan1.head())

          # ?????? burada, names'in altina yazilan "'AMM due to War (2002)'" ifade

               Country  AMM due to War (2002)
0              Albania               0.128908
1              Algeria              18.314120
2              Andorra               0.000000
3               Angola              18.964560
4  Antigua and Barbuda               0.000000
```

```
In [ ]: hakan2 = hakan.parse(1, parse_cols=[0], skiprows=1, names=['Country'])
```

```
print(hakan2.head())

# bu komut calismadi, neredeyse exercise'daki ile ayni ifadeler, anlamadi
```

Importing SAS/Stata files using pandas

In [ ]:

```
# How to import SAS7BDAT
# from sas7bdat import SAS7BDAT
```

In [175]:

```
import pandas as pd
```

In [185]:

```
import matplotlib.pyplot as plt
```

In [ ]:

```
from sas7bdat import SAS7BDAT
```

In [ ]:

```
with SAS7BDAT('sales.sas7bdat') as file
    df_sas = file.to_data_frame()
print(df_sas.head())

# Plot histogram of DataFrame features (pandas and pyplot already importe
pd.DataFrame.hist(df_sas[['P']])
plt.ylabel('count')
plt.show()

# SAS7BDAT ile ilgii komutu kabul etmedigi icin calistiramadim.
```

import Stata files

In [ ]:

```
df = pd.read_stata('disarea.dta')
print(df.head())

# Plot histogram of one column of the DataFrame
pd.DataFrame.hist(df[['disa10']])
plt.xlabel('Extent of disease')
plt.ylabel('Number of coutries')
plt.show()
```

# Using h5py to import HDF5 files

In [ ]:

```
# Import packages
import numpy as np
import h5py

# Assign filename: file
file = 'LIGO_data.hdf5'
```

```
# Load file: data
data = h5py.File(file, 'r')

# Print the datatype of the loaded file
print(type(data))

# Print the keys of the file
for key in data.keys():
    print(key)

<class 'h5py._hl.files.File'>
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
```

# Extracting data from your HDF5 file

In [ ]:

```
# Get the HDF5 group: group
group = data['strain']

# Check out keys of group
for key in group.keys():
    print(key)

# Set variable equal to time series data: strain
strain = data['strain']['Strain'].value

# Set number of time points to sample: num_samples
num_samples = 10000

# Set time vector
time = np.arange(0, 1, 1/num_samples)

# Plot data
plt.plot(time, strain[:num_samples])
plt.xlabel('GPS Time (s)')
plt.ylabel('strain')
plt.show()
```

Loading .mat files

In [ ]:

```
import scipy.io

mat = scipy.io.loadmat('albeck_gene_expression.mat')

# scipy.io.savemat -> write mat files

print(type(mat))
<class 'dict'>
```

The structure of .mat in Python

In [ ]:

```
import scipy.io
import matplotlib.pyplot as plt
import numpy as np

# Print the keys of the MATLAB dictionary
for key in mat.keys():
    print(key)

# ya da dogrudan "print(mat.keys())"

# Print the type of the value corresponding to the key 'CYratioCyt'
print(type(mat['CYratioCyt']))
```

```
# Print the shape of the value corresponding to the key 'CYratioCyt'
print(np.shape(mat['CYratioCyt']))

# Subset the array and plot it
data = mat['CYratioCyt'][25, 5:]
fig = plt.figure()
plt.plot(data)
plt.xlabel('time (min.)')
plt.ylabel('normalized fluorescence (measure of expression)')
plt.show()
```

Relational Databases

```
In [190]: from sqlalchemy import create_engine
```

```
In [191]: engine = create_engine('sqlite:///Chinook.sqlite')
```

```
In [193]: table_names = engine.table_names()
          print(table_names)
```

```
['Album', 'Artist', 'Customer', 'Employee', 'Genre', 'Invoice', 'Invoi
ceLine', 'MediaType', 'Playlist', 'PlaylistTrack', 'Track']
```

```
In [ ]: # Workflow of SQL querying
        • Import packages and functions
        • Create the database engine
        • Connect to the engine
        • Query the database
        • Save query results to a DataFrame
        • Close the connection

        In [1]: from sqlalchemy import create_engine
        In [2]: import pandas as pd
        In [3]: engine = create_engine('sqlite:///Northwind.sqlite')
        In [4]: con = engine.connect()
        In [5]: rs = con.execute("SELECT * FROM Orders")
        In [6]: df = pd.DataFrame(rs.fetchall())
        In [7]: df.columns = rs.keys()
        In [8]: con.close()

        # Using the Context Manager
        In [1]: from sqlalchemy import create_engine
        In [2]: import pandas as pd
        In [3]: engine = create_engine('sqlite:///Northwind.sqlite')
        In [4]: with engine.connect() as con:
           ...: rs = con.execute("SELECT OrderID, OrderDate, ShipName FROM Orders
           ...: df = pd.DataFrame(rs.fetchmany(size=5))
           ...: df.columns = rs.keys()
```

Customizing the Hello World of SQL Queries

```
In [ ]: with engine.connect() as con:
            rs = con.execute("SELECT LastName, Title FROM Employee")
            df = pd.DataFrame(rs.fetchmany(size=3))
            df.columns = rs.keys()

        print(len(df))
        print(df.head())

        #
        3
          LastName                Title
        0    Adams      General Manager
        1  Edwards        Sales Manager
        2  Peacock  Sales Support Agent
```

Filtering your database records using SQL's WHERE

In [ ]:

```python
engine = create_engine('sqlite:///Chinook.sqlite')

with engine.connect() as con:
    rs = con.execute("SELECT * FROM Employee WHERE EmployeeId >= 6")
    df = pd.DataFrame(rs.fetchall())
    df.columns = rs.keys()

print(df.head())
```

Ordering your SQL records with ORDER BY

In [ ]:

```python
engine = create_engine('sqlite:///Chinook.sqlite')

with engine.connect() as con:
    rs = con.execute("SELECT * FROM Employee ORDER BY BirthDate")
    df = pd.DataFrame(rs.fetchall())

df.columns = rs.keys()

print(df.head())
```

Querying relational databases directly with pandas

In [ ]:

```python
In [5]: df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

In [ ]:

```python
# Import packages
from sqlalchemy import create_engine
import pandas as pd

# Create engine: engine
engine = create_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df
df = pd.read_sql_query("SELECT * FROM Album", engine)

# Print head of DataFrame
print(df.head())

# Open engine in context manager
# Perform query and save results to DataFrame: df1
with engine.connect() as con:
    rs = con.execute("SELECT * FROM Album")
    df1 = pd.DataFrame(rs.fetchall())
    df1.columns = rs.keys()

# Confirm that both methods yield the same result: does df = df1 ?
print(df.equals(df1))
```

```
   AlbumId                                  Title  ArtistId
0        1  For Those About To Rock We Salute You         1
1        2                      Balls to the Wall         2
2        3                      Restless and Wild         2
3        4                      Let There Be Rock         1
4        5                               Big Ones         3
True
```

Pandas for more complex querying

```
In [ ]: df = pd.read_sql_query("SELECT * FROM Employee WHERE EmployeeId >= 6 ORDE
```

Advanced querying: exploiting table relationships

```
In [ ]: In [1]: from sqlalchemy import create_engine
        In [2]: import pandas as pd
        In [3]: engine = create_engine('sqlite:///Northwind.sqlite')
        In [4]: df = pd.read_sql_query("SELECT OrderID, CompanyName FROM Orders I
                                       Orders.CustomerID = Customers.CustomerID",
        In [5]: print(df.head())
        OrderID CompanyName
        0 10248 Vins et alcools Chevalier
        1 10251 Victuailles en stock
        2 10254 Chop-suey Chinese
        3 10256 Wellington Importadora
        4 10258 Ernst Handel
```

```
In [ ]: # Open engine in context manager
        # Perform query and save results to DataFrame: df
        with engine.connect() as con:
            rs = con.execute("SELECT Title, Name FROM Album INNER JOIN Artist on
            df = pd.DataFrame(rs.fetchall())
            df.columns = rs.keys()

        # Print head of DataFrame df
        print(df.head())

                                           Title       Name
        0  For Those About To Rock We Salute You      AC/DC
        1                      Balls to the Wall     Accept
        2                      Restless and Wild     Accept
        3                      Let There Be Rock      AC/DC
        4                               Big Ones  Aerosmith
```

```
In [ ]: df = pd.read_sql_query("SELECT * FROM PlaylistTrack INNER JOIN Track on
             PlaylistTrack.TrackId = Track.TrackId WHERE Milliseconds < 250000",
```