Chapter 1)

1)

```
# Open a file: file
file = open('moby_dick.txt', mode='r')

# Print it
print(file.read())

# Check whether file is closed
print(file.closed)
```

```
False
```

```
# Close file
file.close()

# Check whether file is closed
print(file.closed)
```

```
True
```

2)

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())
```

Note)

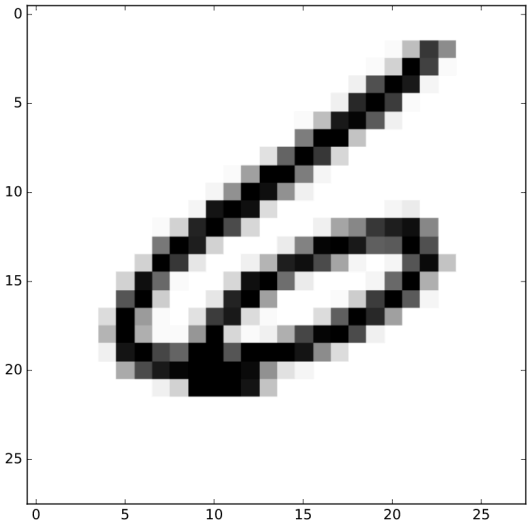
Customizing your NumPy import

```
In [1]: import numpy as np

In [2]: filename = 'MNIST_header.txt'

In [3]: data = np.loadtxt(filename, delimiter=',',
skiprows=1)

In [4]: print(data)
[[   0.    0.    0.    0.    0.]
 [  86.  250.  254.  254.  254.]
 [   0.    0.    0.    9.  254.]
 ...,
 [   0.    0.    0.    0.    0.]
 [   0.    0.    0.    0.    0.]
 [   0.    0.    0.    0.    0.]]
```

```
In [3]: data = np.loadtxt(filename, delimiter=',', skiprows=1,
usecols=[0, 2])

In [4]: print(data)
[[   0.    0.]
 [  86.  254.]
 [   0.    0.]
 ...,
 [   0.    0.]
 [   0.    0.]
 [   0.    0.]]
```

3)

```
# Import package
import numpy as np

# Assign filename to variable: file
file = 'digits.csv'

# Load file as array: digits
digits = np.loadtxt(file, delimiter=',')

# Print datatype of digits
print(type(digits))

# Select and reshape a row
im = digits[21, 1:]
im_sq = np.reshape(im, (28, 28))

# Plot reshaped data (matplotlib.pyplot already loaded as plt)
plt.imshow(im_sq, cmap='Greys', interpolation='nearest')
plt.show()
```



4)
Customizing your NumPy import

What if there are rows, such as a header, that you don't want to import? What if your file has a delimiter other than a comma? What if you only wish to import particular columns?
There are a number of arguments that `np.loadtxt()` takes that you'll find useful: `delimiter` changes the delimiter that `loadtxt()` is expecting, for example, you can use `','` and `'\t'` for comma-delimited and tab-delimited respectively; `skiprows` allows you to specify *how many rows* (not indices) you wish to skip; `usecols` takes a *list* of the indices of the columns you wish to keep.
The file that you'll be importing, `digits_header.txt`,

- has a header
- is tab-delimited.

```
# Import numpy
import numpy as np

# Assign the filename: file
file = 'digits_header.txt'

# Load the data: data
data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2])

# Print data
print(data)
```

5)
Importing different datatypes

The file `seaslug.txt`

- has a text header, consisting of strings
- is tab-delimited.

These data consists of percentage of sea slug larvae that had metamorphosed in a given time period. Read more **here**.
Due to the header, if you tried to import it as-is using `np.loadtxt()`, Python would throw you a `ValueError` and tell you that it `could not convert string to float`. There are two ways to deal with this: firstly, you can set the data type argument `dtype` equal to `str` (for string).

Alternatively, you can skip the first row as we have seen before, using the `skiprows` argument.

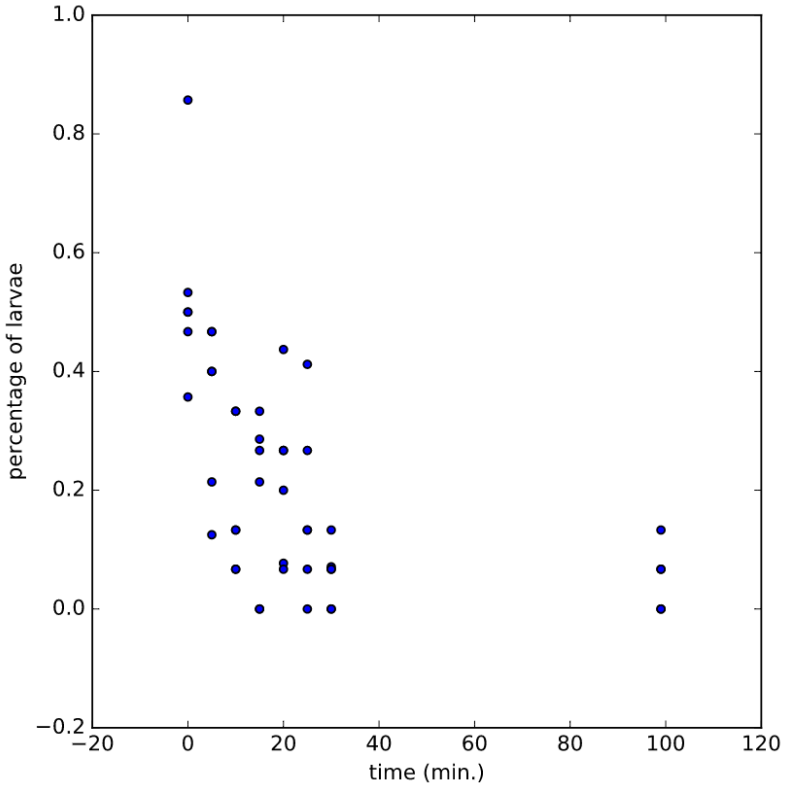
```
# Assign filename: file
file = 'seaslug.txt'

# Import file: data
data = np.loadtxt(file, delimiter='\t', dtype=str)

# Print the first element of data
print(data[0])
```

```
['Time' 'Percent']
```

```
# Import data as floats and skip the first row: data_float
data_float = np.loadtxt(file, delimiter='\t', dtype=float, skiprows=1)

# Print the 10th element of data_float
print(data_float[9])
```

```
[ 0.     0.357]
```

```
# Plot a scatterplot of the data
plt.scatter(data_float[:, 0], data_float[:, 1])
plt.xlabel('time (min.)')
plt.ylabel('percentage of larvae')
plt.show()
```



6)
Working with mixed datatypes (1)

Much of the time you will need to import datasets which have different datatypes in different columns; one column may contain strings and another floats, for example. The function `np.loadtxt()` will freak at this. There is another function, `np.genfromtxt()`, which can handle such structures. If we pass `dtype=None` to it, it will figure out what types each column should be. Import `'titanic.csv'` using the function `np.genfromtxt()` as follows:

```
data = np.genfromtxt('titanic.csv', delimiter=',', names=True, dtype=None)
```

Here, the first argument is the filename, the second specifies the delimiter `,` and the third argument `names` tells us there is a header. Because the data are of different types, `data` is an object called a**structured array**. Because numpy arrays have to contain elements that are all the same type, the structured array solves this by being a 1D array, where each element of the array is a row of the flat file imported. You can test this by checking out the array's shape in the shell by executing `np.shape(data)`.

7)
Working with mixed datatypes (2)

You have just used `np.genfromtxt()` to import data containing mixed datatypes. There is also another function `np.recfromcsv()` that behaves similarly to `np.genfromtxt()`, except that its default `dtype` is `None`. In this exercise, you'll practice using this to achieve the same result.

```
# Assign the filename: file
file = 'titanic.csv'

# Import file using np.recfromcsv: d
d = np.recfromcsv(file,delimiter=',',names=True,dtype=None)

# Print out first three entries of d
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')]
```

8)
Using pandas to import flat files as DataFrames (2)

In the last exercise, you were able to import flat files into a `pandas` DataFrame. As a bonus, it is then straightforward to retrieve the corresponding `numpy` array using the attribute `values`. You'll now have a chance to do this using the MNIST dataset, which is available as `digits.csv`.

```
# Assign the filename: file
file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data
data = pd.read_csv(file,nrows=5,header=None)

# Build a numpy array from the DataFrame: data_array
data_array = np.array(data.values)

# Print the datatype of data_array to the shell
print(type(data_array))
```

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

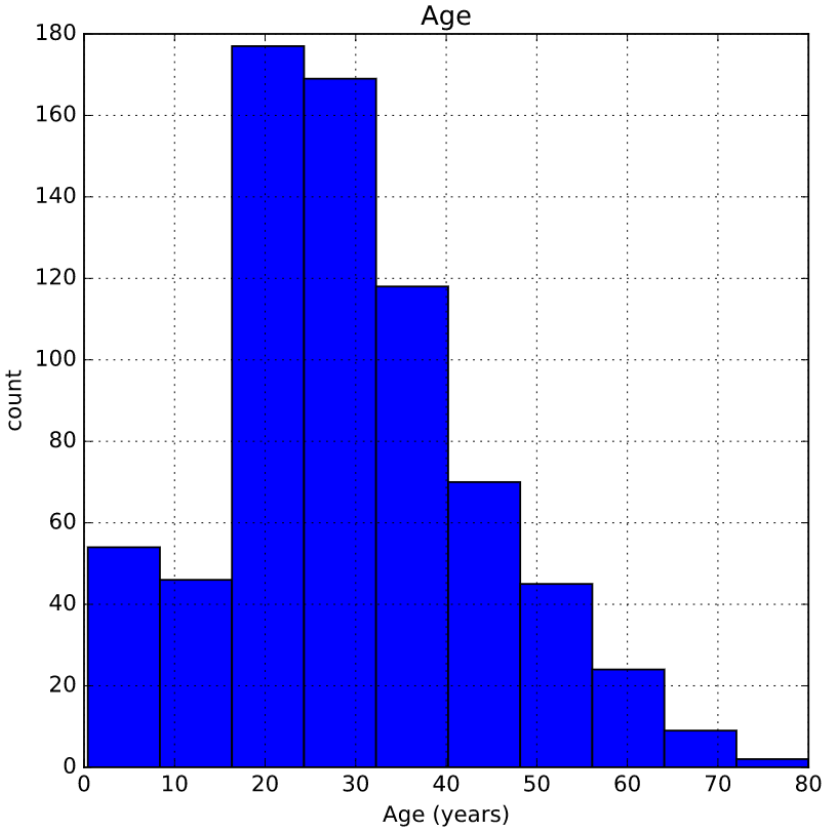
9)
Customizing your pandas import

The `pandas` package is also great at dealing with many of the issues you will encounter when importing data as a data scientist, such as comments occurring in flat files, empty lines and missing values. Note that missing values are also commonly referred to as `NA` or `NaN`. To wrap up this chapter, you're now going to import a slightly corrupted copy of the Titanic dataset `titanic_corrupt.txt`, which

- contains comments after the character `'#'`
- is tab-delimited.

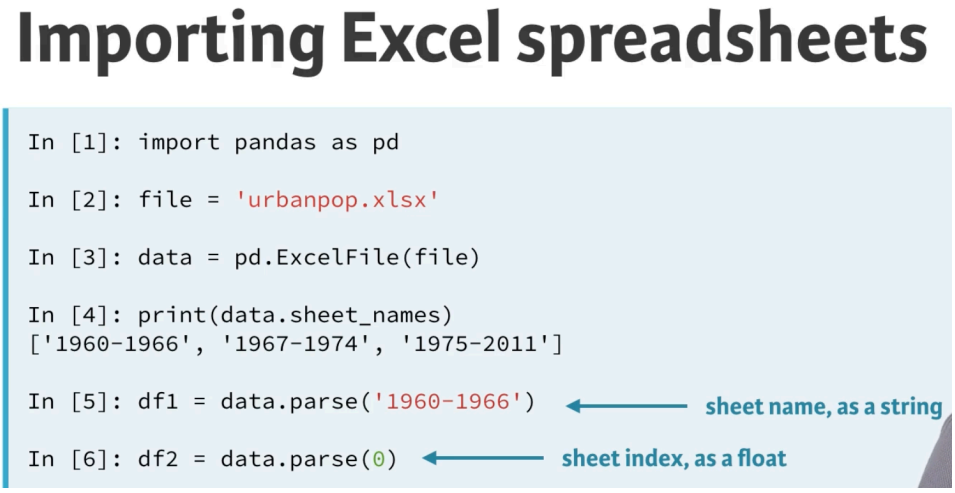
```
# Import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# Assign filename: file
file = 'titanic_corrupt.txt'

# Import file: data
```

```
data = pd.read_csv(file, sep='\t', comment='#', na_values=['Nothing'])

# Print the head of the DataFrame
print(data.head())

# Plot 'Age' variable in a histogram
pd.DataFrame.hist(data[['Age']])
plt.xlabel('Age (years)')
plt.ylabel('count')
plt.show()
```



Chapter 2

1)



2)

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)
```

3)

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 this exercise, you'll import the `pickle` package, open a previously pickled data structure from a file and load it.

```
# Import pickle package
import pickle

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

# Print d
print(d)
```
```
{'Airline': '8', 'Mar': '84.4', 'June': '69.4', 'Aug': '85'}
```
```
# Print datatype of d
print(type(d))
```
```
<class 'dict'>
```

4)

Recall from the video that, given an Excel file imported into a variable `spreadsheet`, you can retrieve a list of the sheet names using the attribute `spreadsheet.sheet_names`.

```
# Import pandas
import pandas as pd

# Assign spreadsheet filename: file
file = 'battledeath.xlsx'

# Load spreadsheet: xl
xl = pd.ExcelFile(file)

# Print sheet names
print(xl.sheet_names)
```

5)

Importing sheets from Excel files

In the previous exercises, you saw that the Excel file contains two sheets, `'2002'` and `'2004'`. The next step is to import these.
In this exercise, you'll learn how to import any given sheet of your loaded .xslx file as a DataFrame. You'll be able to do so by specifying either the sheet's name or its index.
The spreadsheet `'battledeath.xlsx'` is already loaded as `xl`.

```
# Load a sheet into a DataFrame by name: df1
```

```
df1 = xl.parse('2004')

# Print the head of the DataFrame df1
print(df1.head())

# Load a sheet into a DataFrame by index: df2
df2 = xl.parse(0)

# Print the head of the DataFrame df2
print(df2.head())
```

6)

Customizing your spreadsheet import

Here, you'll parse your spreadsheets and use additional arguments to skip rows, rename columns and select only particular columns.
The spreadsheet `'battledeath.xlsx'` is already loaded as `xl`.
As before, you'll use the method `parse()`. This time, however, you'll add the additional arguments `skiprows`, `names` and `parse_cols`. These skip rows, name the columns and designate which columns to parse, respectively. All these arguments can be assigned to lists containing the specific row numbers, strings and column numbers, as appropriate.

```
# Parse the first sheet and rename the columns: df1
df1 = xl.parse(0, skiprows=[0], names=['Country','AAM due to War (2002)'])

# Print the head of the DataFrame df1
print(df1.head())
```

```
              Country  AAM 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
```

```
# Parse the first column of the second sheet and rename the column: df2
df2 = xl.parse(1, parse_cols=[0], skiprows=[0], names=['Country'])

# Print the head of the DataFrame df2
print(df2.head())
```

```
         Country
0        Albania
1        Algeria
2        Andorra
3         Angola
4  Antigua and Barbuda
```

Note)

Importing SAS files

```
In [1]: import pandas as pd

In [2]: from sas7bdat import SAS7BDAT

In [3]: with SAS7BDAT('urbanpop.sas7bdat') as file:
   ...:     df_sas = file.to_data_frame()
```

Importing Stata files

```
In [1]: import pandas as pd

In [2]: data = pd.read_stata('urbanpop.dta')
```

7)

Importing SAS files

In this exercise, you'll figure out how to import a SAS file as a DataFrame using `SAS7BDAT` and `pandas`. The file `'sales.sas7bdat'` is already in your working directory and both `pandas` and `matplotlib.pyplot` have already been imported as follows:

```
import pandas as pd
import matplotlib.pyplot as plt
```

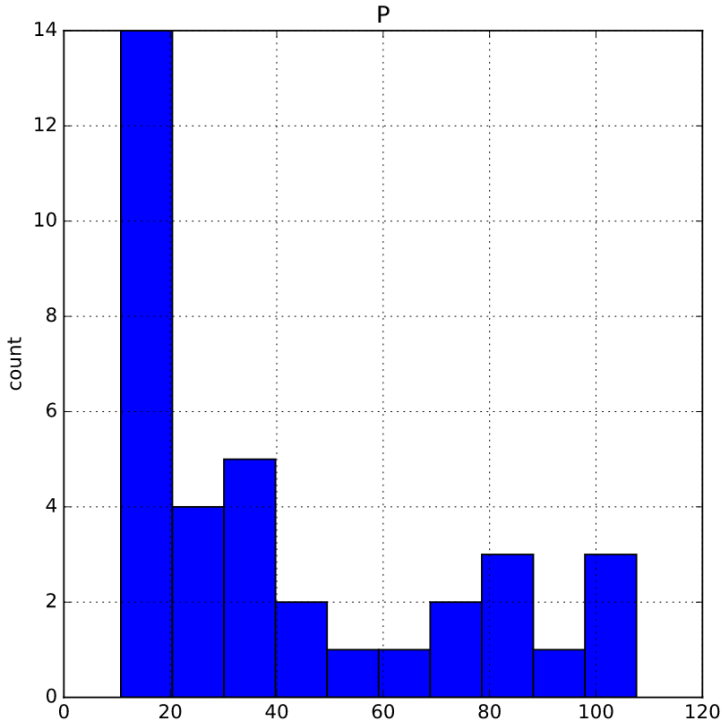
The data are adapted from the website of the undergraduate text book**Principles of Economics** by Hill, Griffiths and Lim.

```
# Import sas7bdat package
from sas7bdat import SAS7BDAT

# Save file to a DataFrame: df_sas
with SAS7BDAT('sales.sas7bdat') as file:
    df_sas = file.to_data_frame()
```

```
# Print head of DataFrame
print (df_sas.head())
```

```
   YEAR    P        S
0  1950.0  12.9  181.899994
1  1951.0  11.9  245.000000
2  1952.0  10.7  250.199997
3  1953.0  11.3  265.899994
4  1954.0  11.2  248.500000
```

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



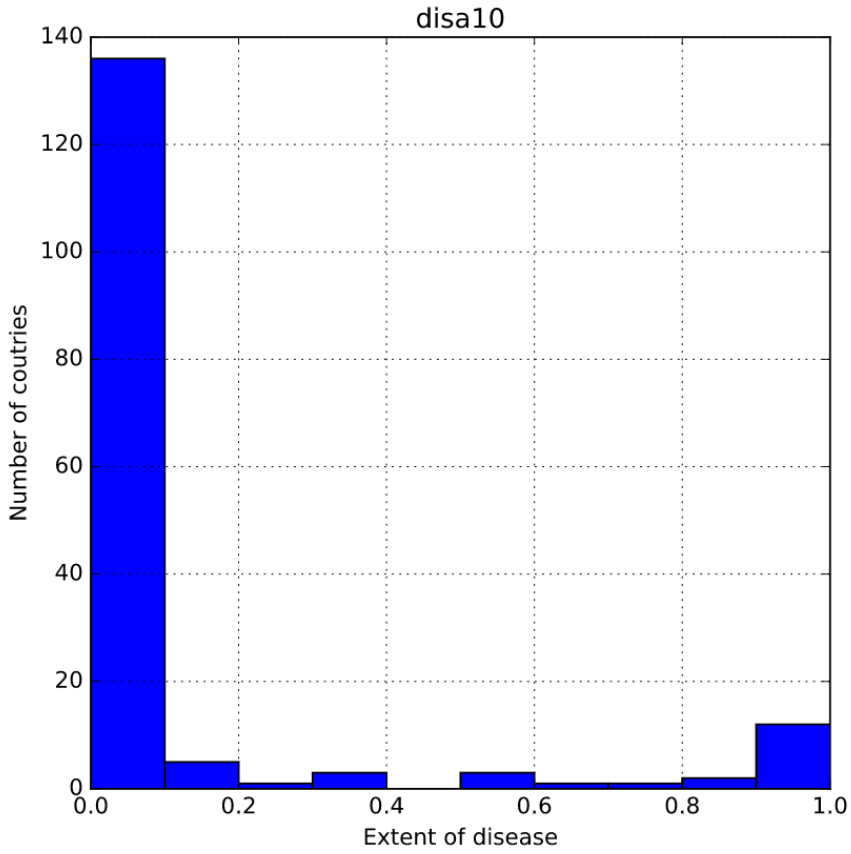
8)

Importing Stata files

Here, you'll gain expertise in importing Stata files as DataFrames using the `pd.read_stata()` function from `pandas`. The last exercise's file, `'disarea.dta'`, is still in your working directory.

```
# Import pandas
import pandas as pd

# Load Stata file into a pandas DataFrame: df
```

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

# Print the head of the DataFrame df
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()
```



Note)

HDF5 files

- Hierarchical Data Format version 5
- Standard for storing large quantities of numerical data
- Datasets can be hundreds of gigabytes or terabytes
- HDF5 can scale to exabytes

Importing HDF5 files

```
In [1]: import h5py

In [2]: filename = 'H-H1_LOSC_4_V1-815411200-4096.hdf5'

In [3]: data = h5py.File(filename, 'r') # 'r' is to read

In [4]: print(type(data))
<class 'h5py._hl.files.File'>
```

The structure of HDF5 files

```
In [5]: for key in data.keys():
   ...:     print(key)
meta
quality
strain

In [6]: print(type(data['meta']))
<class 'h5py._hl.group.Group'>
```

The structure of HDF5 files

```
In [7]: for key in data['meta'].keys():
   ...:         print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart

In [8]: print(data['meta']['Description'].value, data['meta']
['Detector'].value)
b'Strain data time series from LIGO' b'H1'
```

9)

Using h5py to import HDF5 files

The file `'LIGO_data.hdf5'` is already in your working directory. In this exercise, you'll import it using the `h5py` library. You'll also print out its datatype to confirm you have imported it correctly. You'll then study the structure of the file in order to see precisely what HDF groups it contains.
You can find the LIGO data plus loads of documentation and tutorials here.
There is also a great tutorial on Signal Processing with the data here.

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)
```

10)

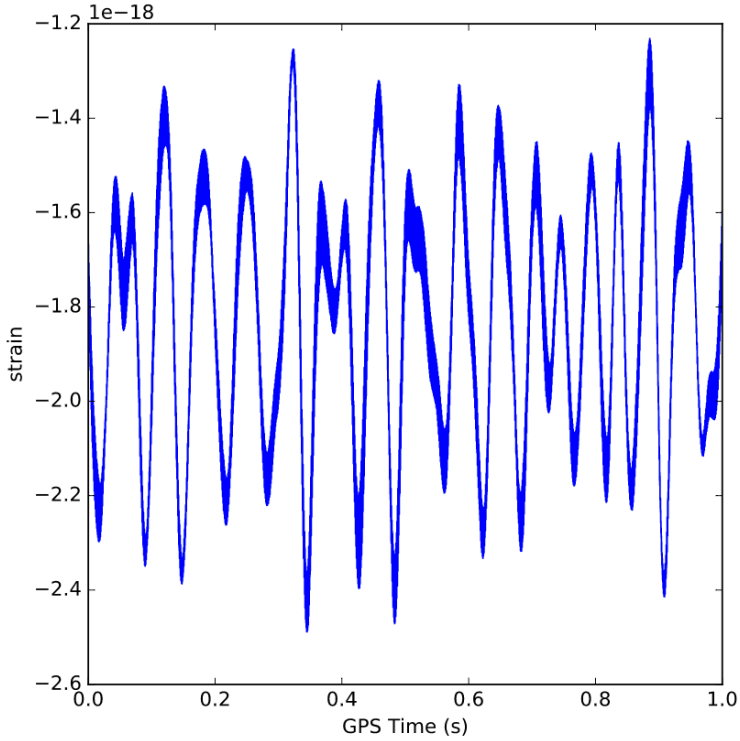
Extracting data from your HDF5 file

In this exercise, you'll extract some of the LIGO experiment's actual data from the HDF5 file and you'll visualize it.
To do so, you'll need to first explore the HDF5 group `'strain'`.

- Assign the HDF5 group `data['strain']` to `group`.
- In the `for` loop, print out the keys of the HDF5 group in `group`.
- Assign to the variable `strain` the values of the time series data `data['strain']['Strain']` using the attribute `.value`.
- Set `num_samples` equal to `10000`, the number of time points we wish to sample.
- Execute the rest of the code to produce a plot of the time series data in `LIGO_data.hdf5`

```
# 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()
```



Note)

SciPy to the rescue!

- `scipy.io.loadmat()` - read .mat files
- `scipy.io.savemat()` - write .mat files

Importing a .mat file

```
In [1]: import scipy.io

In [2]: filename = 'workspace.mat'

In [3]: mat = scipy.io.loadmat(filename)

In [4]: print(type(mat))
<class 'dict'>
```

- keys = MATLAB variable names
- values = objects assigned to variables

11)
Loading .mat files

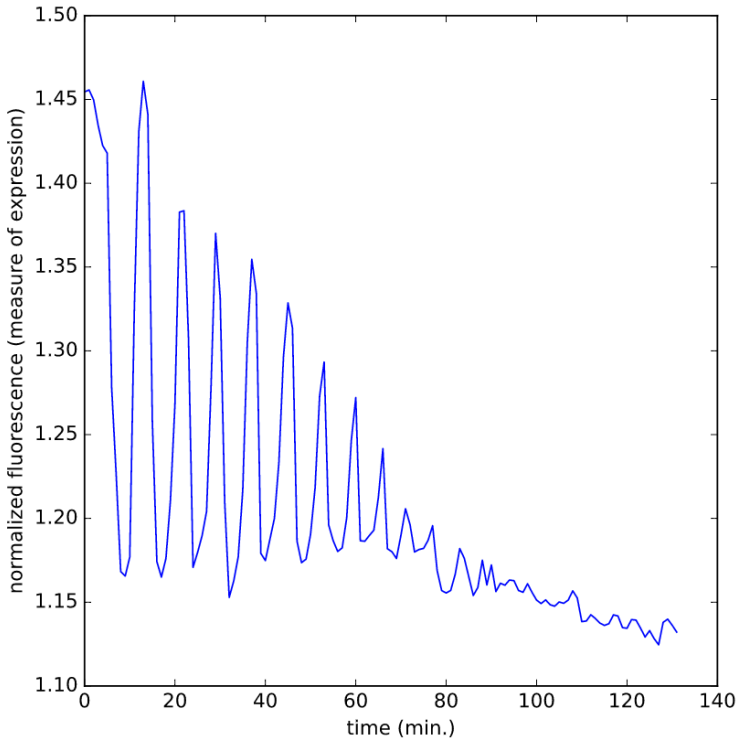
In this exercise, you'll figure out how to load a MATLAB file using `scipy.io.loadmat()` and you'll discover what Python datatype it yields. The file `'albeck_gene_expression.mat'` is in your working directory. This file contains **gene expression data** from the Albeck Lab at UC Davis. You can find the data and some great documentation **here**.

```
# Import package
import scipy.io

# Load MATLAB file: mat
mat = scipy.io.loadmat('albeck_gene_expression.mat')

# Print the datatype type of mat
print(type(mat))

# Print the keys of the MATLAB dictionary
print (mat.keys())
```

dict_keys(['CYratioCyt', '__globals__', 'cfpCyt', '__version__', 'cfpNuc', 'yfpCyt', 'rfpNuc', 'rfpCyt', '__header__', 'yfpNuc'])

```
# Print the type of the value corresponding to the key 'CYratioCyt'
```

```
print (type(mat['CYratioCyt']))
```
```
<class 'numpy.ndarray'>
```
```
# Print the shape of the value corresponding to the key 'CYratioCyt'
print (np.shape(mat['CYratioCyt']))
```
```
(200, 137)
```
```
# 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()
```



Chapter 3)

Note)

Getting table names

```
In [1]: from sqlalchemy import create_engine

In [2]: engine = create_engine('sqlite:///Northwind.sqlite')
```

```
In [3]: table_names = engine.table_names()

In [4]: print(table_names)
['Categories', 'Customers', 'EmployeeTerritories',
'Employees', 'Order Details', 'Orders', 'Products',
'Region', 'Shippers', 'Suppliers', 'Territories']
```

1)

What are the tables in the database?

In this exercise, you'll once again create an engine to connect to `'Chinook.sqlite'`. Before you can get any data out of the database, however, you'll need to know what tables it contains!
To this end, you'll save the table names to a list using the method `table_names()` on the engine and then you will print the list.

Creating a database engine

Here, you're going to fire up your very first SQL engine. You'll create an engine to connect to the SQLite database `'Chinook.sqlite'`, which is in your working directory. Remember that to create an engine to connect to `'Northwind.sqlite'`, Hugo executed the command

```
engine = create_engine('sqlite:///Northwind.sqlite')
```

Here, `'sqlite:///Northwind.sqlite'` is called the *connection string* to the SQLite database `Northwind.sqlite`. A little bit of background on the **Chinook database**: the Chinook database contains information about a semi-fictional digital media store in which media data is real and customer, employee and sales data has been manually created.

# Import necessary module
from sqlalchemy import create_engine

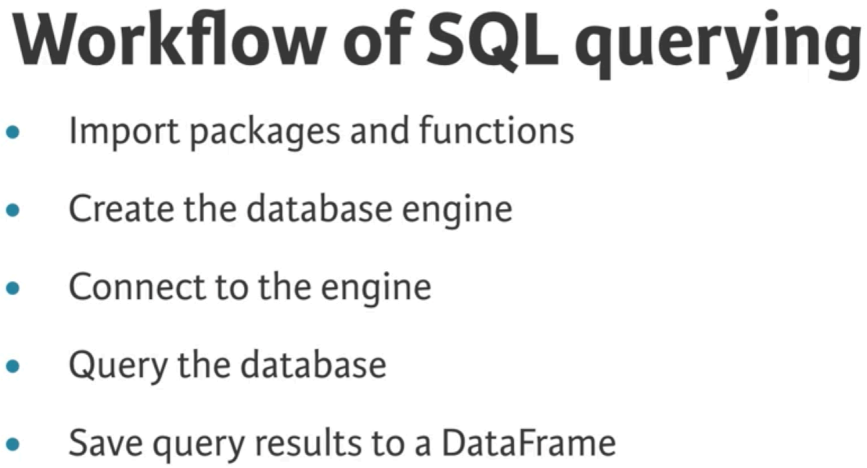
```
# Create engine: engine
engine = create_engine('sqlite:///Chinook.sqlite')
```

2)

```
# Import necessary module
from sqlalchemy import create_engine

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

# Save the table names to a list: table_names
table_names = engine.table_names()

# Print the table names to the shell
print(table_names)
```

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

Note)



Your first SQL query

```
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]: con.close()
```

Set the DataFrame column names

```
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()
```

3)

The Hello World of SQL Queries!

Now, it's time for liftoff! In this exercise, you'll perform the Hello World of SQL queries, `SELECT`, in order to retrieve all columns of the table `Album` in the Chinook database. Recall that the query `SELECT *` selects all columns.

- Open the engine connection as `con` using the method `connect()` on the engine.
- Execute the query that **selects** *ALL columns* **from** the `Album` table. Store the results in `rs`.
- Store all of your query results in the DataFrame `df` by applying the `fetchall()` method to the results `rs`.

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

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

# Open engine connection: con
con = engine.connect()

# Perform query: rs
rs = con.execute('Select * from Album')

# Save results of the query to DataFrame: df
df = pd.DataFrame(rs.fetchall())

# Close connection
con.close()

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

```
   0                                  1  2
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
```

4)
Customizing the Hello World of SQL Queries

Congratulations on executing your first SQL query! Now you're going to figure out how to customize your query in order to:

- Select specified columns from a table;
- Select a specified number of rows;
- Import column names from the database table.

```
# Open engine in context manager
# Perform query and save results to DataFrame: df
with engine.connect() as con:
    rs = con.execute('Select LastName,Title from Employee')
    df = pd.DataFrame(rs.fetchmany(3))
    df.columns = rs.keys()

# Print the length of the DataFrame df
print(len(df))

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

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

5)

There are a couple more standard SQL query chops that will aid you in your journey to becoming an SQL ninja.

Let's say, for example that you wanted to get all records from the `Customer`table of the Chinook database for which the `Country` is `'Canada'`. You can do this very easily in SQL using a `SELECT` statement followed by a `WHERE` clause as follows:

```
SELECT * FROM Customer WHERE Country = 'Canada'
```

In fact, you can filter any `SELECT` statement by any condition using a `WHERE`clause. This is called *filtering* your records.

In this interactive exercise, you'll select all records of the `Employee` table for which `'EmployeeId'` is greater than or equal to `6`.

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

# Open engine in context manager
# Perform query and save results to DataFrame: df
```

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

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

```
   EmployeeId  LastName FirstName       Title  ReportsTo           BirthDate  \
0           6  Mitchell   Michael  IT Manager           1 1973-07-01 00:00:00
1           7      King    Robert    IT Staff           6 1970-05-29 00:00:00
2           8  Callahan     Laura    IT Staff           6 1968-01-09 00:00:00

             HireDate                  Address       City State Country  \
0 2003-10-17 00:00:00       5827 Bowness Road NW     Calgary    AB  Canada
1 2004-01-02 00:00:00 590 Columbia Boulevard West  Lethbridge    AB  Canada
2 2004-03-04 00:00:00               923 7 ST NW  Lethbridge    AB  Canada

  PostalCode           Phone             Fax                 Email
0    T3B 0C5  +1 (403) 246-9887  +1 (403) 246-9899  michael@chinookcorp.com
1    T1K 5N8  +1 (403) 456-9986  +1 (403) 456-8485   robert@chinookcorp.com
2    T1H 1Y8  +1 (403) 467-3351  +1 (403) 467-8772    laura@chinookcorp.com
```

6)

Ordering your SQL records with ORDER BY

You can also *order* your SQL query results. For example, if you wanted to get all records from the `Customer` table of the Chinook database and order them in increasing order by the column `SupportRepId`, you could do so with the following query:

```
"SELECT * FROM Customer ORDER BY SupportRepId"
```

In fact, you can order any `SELECT` statement by any column.

In this interactive exercise, you'll select all records of the `Employee` table and order them in increasing order by the column `BirthDate`.

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

# Open engine in context manager
with engine.connect() as con:
    rs = con.execute('SELECT * from Employee ORDER BY BirthDate')
    df = pd.DataFrame(rs.fetchall())

    # Set the DataFrame's column names
    df.columns = rs.keys()
```

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

```
   EmployeeId  LastName FirstName              Title  ReportsTo  \
0           4      Park  Margaret  Sales Support Agent        2.0
1           2   Edwards     Nancy        Sales Manager        1.0
2           1     Adams    Andrew      General Manager        NaN
3           5   Johnson     Steve  Sales Support Agent        2.0
4           8  Callahan     Laura           IT Staff        6.0

             BirthDate            HireDate              Address       City  \
0  1947-09-19 00:00:00  2003-05-03 00:00:00     683 10 Street SW    Calgary
1  1958-12-08 00:00:00  2002-05-01 00:00:00        825 8 Ave SW    Calgary
2  1962-02-18 00:00:00  2002-08-14 00:00:00  11120 Jasper Ave NW   Edmonton
3  1965-03-03 00:00:00  2003-10-17 00:00:00        7727B 41 Ave    Calgary
4  1968-01-09 00:00:00  2004-03-04 00:00:00         923 7 ST NW  Lethbridge

  State Country PostalCode            Phone              Fax  \
0    AB  Canada    T2P 5G3  +1 (403) 263-4423  +1 (403) 263-4289
1    AB  Canada    T2P 2T3  +1 (403) 262-3443  +1 (403) 262-3322
2    AB  Canada    T5K 2N1  +1 (780) 428-9482  +1 (780) 428-3457
3    AB  Canada    T3B 1Y7    1 (780) 836-9987    1 (780) 836-9543
4    AB  Canada    T1H 1Y8  +1 (403) 467-3351  +1 (403) 467-8772

                  Email
0  margaret@chinookcorp.com
1     nancy@chinookcorp.com
2    andrew@chinookcorp.com
3     steve@chinookcorp.com
4     laura@chinookcorp.com
```

Note)

The pandas way to query

```
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 * FROM Orders")
   ...:     df = pd.DataFrame(rs.fetchall())
   ...:     df.columns = rs.keys()
```

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

7)

Pandas and The Hello World of SQL Queries!

Here, you'll take advantage of the power of `pandas` to write the results of your SQL query to a DataFrame in one swift line of Python code!
You'll first import `pandas` and create the SQLite `'Chinook.sqlite'` engine. Then you'll query the database to select all records from the `Album` table.
Recall that to select all records from the `Orders` table in the Northwind database, Hugo executed the following command:

```
df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

```
# 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())
```

```
   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
```

```
# 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))
```

```
True
```

8)

Pandas for more complex querying

Here, you'll become more familiar with the pandas function `read_sql_query()` by using it to execute a more complex query: a `SELECT` statement followed by both a `WHERE` clause AND an `ORDER BY` clause.
You'll build a DataFrame that contains the rows of the `Employee` table for which the `EmployeeId` is greater than or equal to `6` and you'll order these entries by `BirthDate`.

```
# 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 Employee WHERE EmployeeId>=6
ORDER BY BirthDate',engine)

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

```
   EmployeeId  LastName FirstName       Title  ReportsTo           BirthDate  \
0           6  Mitchell   Michael  IT Manager           1  1973-07-01 00:00:00
1           7      King    Robert    IT Staff           6  1970-05-29 00:00:00
2           8  Callahan     Laura    IT Staff           6  1968-01-09 00:00:00

             HireDate                   Address       City State Country  \
0  2003-10-17 00:00:00        5827 Bowness Road NW     Calgary    AB  Canada
1  2004-01-02 00:00:00  590 Columbia Boulevard West  Lethbridge    AB  Canada
2  2004-03-04 00:00:00               923 7 ST NW  Lethbridge    AB  Canada

  PostalCode            Phone             Fax                 Email
0    T3B 0C5  +1 (403) 246-9887  +1 (403) 246-9899  michael@chinookcorp.com
1    T1K 5N8  +1 (403) 456-9986  +1 (403) 456-8485   robert@chinookcorp.com
2    T1H 1Y8  +1 (403) 467-3351  +1 (403) 467-8772    laura@chinookcorp.com

<script.py> output:
      EmployeeId  LastName FirstName       Title  ReportsTo           BirthDate  \
   0           8  Callahan     Laura    IT Staff           6  1968-01-09 00:00:00
   1           7      King    Robert    IT Staff           6  1970-05-29 00:00:00
   2           6  Mitchell   Michael  IT Manager           1  1973-07-01 00:00:00

             HireDate                   Address       City State Country  \
   0  2004-03-04 00:00:00               923 7 ST NW  Lethbridge    AB  Canada
   1  2004-01-02 00:00:00  590 Columbia Boulevard West  Lethbridge    AB  Canada
   2  2003-10-17 00:00:00        5827 Bowness Road NW     Calgary    AB  Canada

    PostalCode            Phone             Fax                 Email
   0    T1H 1Y8  +1 (403) 467-3351  +1 (403) 467-8772    laura@chinookcorp.com
   1    T1K 5N8  +1 (403) 456-9986  +1 (403) 456-8485   robert@chinookcorp.com
   2    T3B 0C5  +1 (403) 246-9887  +1 (403) 246-9899  michael@chinookcorp.com
```

Note)

9)

The power of SQL lies in relationships between tables: INNER JOIN

Here, you'll perform your first `INNER JOIN`! You'll be working with your favourite SQLite database, `Chinook.sqlite`. For each record in the `Album`table, you'll extract the `Title` along with the `Name` of the `Artist`. The latter will come from the `Artist` table and so you will need to `INNER JOIN`these two tables on the `ArtistID` column of both.
Recall that to `INNER JOIN` the `Orders` and `Customers` tables from the Northwind database, Hugo executed the following SQL query:

```
"SELECT OrderID, CompanyName FROM Orders INNER JOIN Customers on
Orders.CustomerID = Customers.CustomerID"
```

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

10)

```
# Execute query and store records in DataFrame: df
df = pd.read_sql_query('SELECT * FROM PlaylistTrack INNER JOIN Track
on PlaylistTrack.TrackId=Track.TrackId WHERE Milliseconds <
250000',engine)

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