Exercise1

Importing entire text files

In this exercise, you'll be working with the file `moby_dick.txt` . It is a text file that contains the opening sentences of Moby Dick, one of the great American novels! Here you'll get experience opening a text file, printing its contents to the shell and, finally, closing it.

Instructions

- Open the file moby_dick.txt as read-only and store it in the variable file. Make sure to pass the filename enclosed in quotation marks ".
- Print the contents of the file to the shell using the print() function. As Hugo showed in the video, you'll need to apply the method read() to the object file.
- Check whether the file is closed by executing print(file.closed).
- Close the file using the close() method.
- Check again that the file is closed as you did above.

In [ ]:

```
# Open a file: file
file = open(____, ____)

# Print it
print(____)

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

# Close file

# Check whether file is closed

#_________________#
#Solutions

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

# Print it
print(file.read())

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

# Close file
file.close()

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

Exercise2

Importing text files line by line

For large files, we may not want to print all of their content to the shell: you may wish to print only the first few lines. Enter the readline() method, which allows you to do this. When a file called file is open, you can print out the first line by executing file.readline(). If you execute the same command again, the second line will print, and so on.

In the introductory video, Hugo also introduced the concept of a context manager. He showed that you can bind a variable file by using a context manager construct:

```
with open('huck_finn.txt') as file:
```

While still within this construct, the variable file will be bound to open('huck_finn.txt'); thus, to print the file to the shell, all the code you need to execute is:

```
with open('huck_finn.txt') as file:
    print(file.readline())
```

You'll now use these tools to print the first few lines of `moby_dick.txt` !

Instructions

- Open moby_dick.txt using the with context manager and the variable file.
- Print the first three lines of the file to the shell by using readline() three times within the context manager.

In [ ]:

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

#_________________#
#Solutions

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

Exercise3

Using NumPy to import flat files

In this exercise, you're now going to load the MNIST digit recognition dataset using the numpy function loadtxt() and see just how easy it can be:

- The first argument will be the filename.
- The second will be the delimiter which, in this case, is a comma.

You can find more information about the MNIST dataset here on the webpage of Yann LeCun, who is currently Director of AI Research at Facebook and Founding Director of the NYU Center for Data Science, among many other things.

Instructions

- Fill in the arguments of np.loadtxt() by passing file and a comma ',' for the delimiter.
- Fill in the argument of print() to print the type of the object digits. Use the function type().
- Execute the rest of the code to visualize one of the rows of the data.

In [ ]:

```
# Import package
import numpy as np

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

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

# Print datatype of digits
print(____)

# 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()



#__________________#
#Solutions

# 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()
```

Exercise4

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.
  - You can use ',' for comma-delimited.
  - You can use '\t' for tab-delimited.
- `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 and is tab-delimited.

Instructions

- Complete the arguments of np.loadtxt(): the file you're importing is tab-delimited, you want to skip the first row and you only want to import the first and third columns.
- Complete the argument of the print() call in order to print the entire array that you just imported.

In [ ]:

```
# Import numpy
import numpy as np

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

# Load the data: data
data = np.loadtxt(____, delimiter=____, skiprows=____, usecols=____)

# Print data
print(____)


#__________________#
#Solutions

# 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)
```

Exercise5

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.

Instructions

- Complete the first call to np.loadtxt() by passing file as the first argument.
- Execute `print(data[0])` to print the first element of data.
- Complete the second call to np.loadtxt(). The file you're importing is tab-delimited, the datatype is float, and you want to skip the first row.
- Print the 10th element of data_float by completing the print() command. Be guided by the previous print() call.
- Execute the rest of the code to visualize the data.

In [ ]:

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

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

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

# Import data as floats and skip the first row: data_float
data_float = np.loadtxt(____, delimiter=____, dtype=____, skiprows=____)

# Print the 10th element of data_float
print(____)

# 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()



#__________________#
#Solutions

# 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])

# 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])

# 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()
```

Exercise6

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.

Instructions

- Import titanic.csv using the function np.recfromcsv() and assign it to the variable, d. You'll only need to pass file to it because it has the defaults delimiter=',' and names=True in addition to dtype=None!
- Run the remaining code to print the first three entries of the resulting array d

In [ ]:

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

# Import file using np.recfromcsv: d


# Print out first three entries of d
print(d[:3])



#__________________#
#Solutions

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

# Import file using np.recfromcsv: d
d = np.recfromcsv(file)

# Print out first three entries of d
print(d[:3])
```

Exercise7

Using pandas to import flat files as DataFrames (1)

In the last exercise, you were able to import flat files containing columns with different datatypes as numpy arrays. However, the DataFrame object in pandas is a more appropriate structure in which to store such data and, thankfully, we can easily import files of mixed data types as DataFrames using the pandas functions read_csv() and read_table().

Instructions

- Import the pandas package using the alias pd.
- Read titanic.csv into a DataFrame called df. The file name is already stored in the file object.
- In a print() call, view the head of the DataFrame.

In [ ]:

```
# Import pandas as pd


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

# Read the file into a DataFrame: df
df = pd.read_csv(____)

# View the head of the DataFrame


#_________________#
#Solutions

# Import pandas as pd
import pandas as pd

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

# Read the file into a DataFrame: df
df = pd.read_csv(file)

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

Exercise8

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.

Instructions

- Import the first 5 rows of the file into a DataFrame using the function pd.read_csv() and assign the result to data. You'll need to use the arguments nrows and header (there is no header in this file).
- Build a numpy array from the resulting DataFrame in data and assign to data_array.
- Execute print(type(data_array)) to print the datatype of data_array.

In [ ]:

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

# Read the first 5 rows of the file into a DataFrame: data


# Build a numpy array from the DataFrame: data_array


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


#__________________#
#Solutions

# 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 = data.values

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

Exercise9

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.

Instructions

- Complete the sep (the pandas version of delim), comment and na_values arguments of pd.read_csv(). comment takes characters that comments occur after in the file, which in this case is '#'. na_values takes a list of strings to recognize as NA/NaN, in this case the string 'Nothing'.
- Execute the rest of the code to print the head of the resulting DataFrame and plot the histogram of the 'Age' of passengers aboard the Titanic.

In [ ]:

```
# 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=____, comment=____, na_values=____)

# 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()



#__________________#
#Solutions

# 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()
```