Load data into Pandas via "DFs" which we talked about earlier.

you can "print(df)" to print at the whole data frame

print(df.head) prints from the top of the data

print(df.tail) prints from the bottom 3 rows of the data.

Pandas also allows you to read in excel files like:

"df_xlsx = pd.read_excel('')"

Pandas also allows you to load ~~tab-separated (txt)~~ files. you do this by using the read_csv file, but this time you must specify a delimeter.

→ " df = pd.read_csv(' ', delimeter = '\t'

'\t' = tab

A file will generally come with headers to organize the data.

To read these headers, you can do a print(df.columns)

Once you get those columns, you can get the data under a certain header by: print(df['Name'])

you can also overload this command by passing a list instead, and it will print out all of the requested columns.

If you ever want a specific row, you can use the "iloc" function, which stands for "integer location".

→ "print(df.iloc[1])"
will print all everything in that first row

→ "print(df.iloc[1:4])"
alternatively, you can simply use the "hash" operator to print multiple rows

→ "print(df.iloc[2,1])"

The iloc convention can also be used to identify data in a specific location if you use the syntax above. (row, column)

The easiest way to go row by row and access any sort of data you might want is

"for index, row in df.iterrows():
    print(index, row)"

One additional function is df.loc! This is used for finding data in our dataset that isn't just integer-based.

↳ i.e., more based on textual or numerical information

→ "df.loc[df['Type 1'] == "Fire"]"

The above code accesses only the rows that have fire type, for example.

Another useful dataframe method is the "describe" method which gives us all the stats like mean and standard dev.

"df.describe()"

To manipulate how the data is expunged, you can use "df.sort_values()" and insert the filter (the column).

→ df.sort_values('Name', ascending = False)

would print names in reverse alphabetical order