In [3]:
```
import pandas as pd
```

pandas datatypes

- object- text or mixed numeric or non-numeric values
- int64- integer numbers
- bool- truth/false values
- float64- floating point numbers
- category- finite list of text values
- datetime64- date and time values
- timedelta[ns]- differences between two datetimes

pandas datastructures- particular way of organizing data

- Series
- DataFrame
- DataFrames are easy ways to store information and can be though of as spreadsheets. They are composed of a collection of series.
- Series can be described as the single column of a 2-D array that can store data of any type.
- Therefore, DataFrames are tables that use multiple columns and rows
- Each value in a DataFrame object is associated with a row and a column index.

Series

In [4]:
```
series1 = pd.Series([1,2,3,4])

print(series1)
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

In [5]:
```
# Defining a series object
srs = pd.Series([1,2,3,4,5])

# printing series values
print("The Series values are:")
print(srs.values)

# printing series indexes
print("\nThe Index values are:")
print(srs.index.values)
```

```
The Series values are:
[1 2 3 4 5]
```

```
The Index values are:
[0 1 2 3 4]
```

Assign names to our values

Pandas will automatically generate our indexes, so we need to define them. Each index corresponds to its value in the **Series** object. Let's assign a country name to population growth rates.

In [8]:
```
# One can define Series values and indexes separately like so:
srs = pd.Series([11.9, 36.0, 16.6, 21.8, 34.2], index = ['China', 'India', 'USA', 'Braz

#Set Series name
srs.name = "Growth Rate"

# Set index name
srs.index.name = "Country"

# printing series values
print("The Indexed Series values are:")
print(srs)
```

```
The Indexed Series values are:
Country
China       11.9
India       36.0
USA         16.6
Brazil      21.8
Pakistan    34.2
Name: Growth Rate, dtype: float64
```

In [3]:
```
# To select entries from a Series, we select elements based on the index name or index

import numpy as np
import pandas as pd

srs = pd.Series(np.arange(0, 6, 1), index = ['ind0', 'ind1', 'ind2', 'ind3', 'ind4', 'i
# I am guessing that np.arange is just a range but from NumPy
srs.index.name = "Index"
print(f"The original Series:\n{srs}")

print("\nSeries element at index ind3:")
print(srs['ind3']) # Fetch element at index ind3

print("\nSeries element at index 3:")
print(srs[3]) # Fetch element at index 3

print("\nSeries elements at multiple indexes:\n")
print(srs[['ind1', 'ind4']]) # Fetch elements at multiple indexes
```

```
The original Series:
Index
ind0    0
ind1    1
ind2    2
ind3    3
ind4    4
ind5    5
dtype: int32

Series element at index ind3:
```

```
3

Series element at index 3:
3

Series elements at multiple indexes:

Index
ind1    1
ind4    4
dtype: int32
```

DataFrame: the most important operations

In [6]:

```
df = pd.DataFrame({
    "Column1": [1, 4, 8, 7, 9],
    "Column2": ['a', 'column', 'with', 'a', 'string'],
    "Column3": [1.23, 23.5, 45.6, 32.1234, 89.453],
    "Column4": [True, False, True, False, True]
})
print(df)

# One can also create a dictionary and pass that data to a DataFrame
data = {
    'peppers': [3, 2, 0, 1],
    'carrots': [0, 3, 7, 2]
}

quantity = pd.DataFrame(data)

print()
print(quantity)

# We can also change the index as such
quantity2 = pd.DataFrame(data, index = ['June', 'July', 'August', 'September'])

print(quantity2)
```

```
   Column1 Column2  Column3  Column4
0        1       a   1.2300     True
1        4  column  23.5000    False
2        8    with  45.6000     True
3        7       a  32.1234    False
4        9  string  89.4530     True

   peppers  carrots
0        3        0
1        2        3
2        0        7
3        1        2
           peppers  carrots
June             3        0
July             2        3
August           0        7
September        1        2
```

Other useful operations:

- **.shape**- outputs a tuple of (rows, columns)

- **.columns** shows a dataset's column names
- **.rename()** allows us to rename columns like 'search and replace'

Searching and selecting in a dataframe

- **loc** and **iloc** are used for locating data
- **.iloc[x]** locates by numerical index
- **.loc[sting]** locates by the index name, sumular to list slicing in Python

In [8]:
```
# Easiest way to select a column or multiple columns of data is by using brackets []

df = pd.read_csv('cancer_stats.csv')

print(df.columns) # Print columns of DataFrame

print("\nThe First Column")
print(df['Sex'].head()) # Fetch the sec xolumn from DataFrame
print(f"\nThe type of this column is {str(type(df['Under 1']))}\n")

print("\nThe Last Column")
print(df['40-44'].head()) # Fetch the 40-44 colum from DataFrame
print(f"\nThe type of this column is: {str(type(df['40-44']))}\n")
```

```
Index(['Sex', 'Under 1', '1-4', '5-9', '10-14', '15-19', '20-24', '25-29',
       '30-34', '35-39', '40-44'],
      dtype='object')

The First Column
0      Males
1    Females
2      Males
3    Females
4      Males
Name: Sex, dtype: object

The type of this column is <class 'pandas.core.series.Series'>


The Last Column
0    2045
1    4457
2     139
3      81
4       2
Name: 40-44, dtype: int64

The type of this column is: <class 'pandas.core.series.Series'>
```

Creating a new DataFrame from pre-existing columns

In [9]:
```
df = pd.read_csv('test.csv')

print(df.columns)

print("\nThe original DataFrame:")
print(df.head())
```

```
print("\nThe new DataFrame with selected columns is:\n")
new_df = pd.DataFrame(df, columns=['Sex', 'Under 1', '40-44'])
print(new_df.head())
```

```
Index(['Sex', 'Under 1', '1-4', '5-9', '10-14', '15-19', '20-24', '25-29',
       '30-34', '35-39', '40-44'],
      dtype='object')

The original DataFrame:
       Sex  Under 1  1-4  5-9  10-14  15-19  20-24  25-29  30-34  35-39  40-44
0    Males       82  305  199    197    322    537    910   1239   1610   2045
1  Females       73  249  171    183    297    554   1341   2219   3085   4457
2    Males        0    0    1      4     10      7     17     30     67    139
3  Females        0    1    0      5      8      8     12     29     53     81
4    Males        0    0    0      0      0      0      1      1      1      2

The new DataFrame with selected columns is:

       Sex  Under 1  40-44
0    Males       82   2045
1  Females       73   4457
2    Males        0    139
3  Females        0     81
4    Males        0      2
```

Reindexing data in a DataFrame

We can also reindex the data by either the indexes themselves or the columns. **.reindex()** allows us to make changes w/o messing up the initial setting of the objects. Works the same for both Series and DataFrame objects.

In [10]:

```
srs1 = pd.Series([11.9, 36.0, 16.6, 21.8, 34.2], index = ['China', 'India', 'USA', 'Bra

# Set Series name
srs1.name = "Growth Rate"

# Set index name
srs1.index.name = "Country"

srs2 = srs1.reindex(['China', 'India', 'Malaysia', 'USA', 'Brazil', 'Pakistan', 'Englan
print("The series with new indexes is:\n",srs2)

srs3 = srs1.reindex(['China', 'India', 'Malaysia', 'USA', 'Brazil', 'Pakistan', 'Englan
print("\nThe series with new indexes is:\n",srs3)
```

```
The series with new indexes is:
 Country
China       11.9
India       36.0
Malaysia     NaN
USA         16.6
Brazil      21.8
Pakistan    34.2
England      NaN
Name: Growth Rate, dtype: float64

The series with new indexes is:
 Country
China       11.9
```

```
India       36.0
Malaysia     0.0
USA         16.6
Brazil      21.8
Pakistan    34.2
England      0.0
Name: Growth Rate, dtype: float64
```

Reading and importing data

- For csv:
  - pd.read_csv("file.csv")
  - ps.read_csv("file.csv", index_col = 0)- chooses what column to make the index out of rather than creating an automatic one
  - df.to_csv('file.csv')- saves the dataframe into a csv file
- For JSON:
  - df.read_json('file.json')
  - df.to_json('file.json')
- For Excel:
  - df.read_excel('file.xlsx')

Data Wrangling with Pandas

Data wrangling is basically manipulating and preparing data for analysis.

- Merging
- Concatenation
- Grouping

In [2]:

```
# MERGING

import pandas as pd

d = {
    'subject_id': ['1', '2', '3', '4', '5'],
    'student_name': ['Mark', 'Khalid', 'Deborah', 'Trevon', 'Raven']
}
df1 = pd.DataFrame(d, columns=['subject_id', 'student_name'])
print(df1)

data = {
    'subject_id': ['4', '5', '6', '7', '8'],
    'student_name': ['Eric', 'Imani', 'Cece', 'Darius', 'Andre']
}
df2 = pd.DataFrame(data, columns=['subject_id', 'student_name'])
print(df2)

pd.merge(df1, df2, on='subject_id')
```

```
  subject_id student_name
0          1         Mark
1          2       Khalid
2          3      Deborah
3          4       Trevon
```

```
4           5        Raven
  subject_id student_name
0          4         Eric
1          5        Imani
2          6         Cece
3          7       Darius
4          8        Andre
```

Out[2]:

	subject_id	student_name_x	student_name_y
0	4	Trevon	Eric
1	5	Raven	Imani

In [4]:

```
# GROUPING

raw = {
    'Name': ['Darell', 'Darell', 'Lilith', 'Lilith', 'Tran', 'Tran', 'Tran',
        'Tran', 'John', 'Darell', 'Darell', 'Darell'],
    'Position': [2, 1, 1, 4, 2, 4, 3, 1, 3, 2, 4, 3],
    'Year': [2009, 2010, 2009, 2010, 2010, 2010, 2011, 2012, 2011, 2013, 2013, 2012],
    'Marks':[408, 398, 422, 376, 401, 380, 396, 388, 356, 402, 368, 378]
}
df = pd.DataFrame(raw)

group = df.groupby('Year')
print(group.get_group(2011))
```

```
   Name  Position  Year  Marks
6  Tran         3  2011    396
8  John         3  2011    356
```

In [5]:

```
# CONCATENATION

print(pd.concat([df1, df2]))
```

```
  subject_id student_name
0          1         Mark
1          2       Khalid
2          3      Deborah
3          4       Trevon
4          5        Raven
0          4         Eric
1          5        Imani
2          6         Cece
3          7       Darius
4          8        Andre
```

Other common data wrangling processes you should know:

- Mapping data and finding duplicates
- Finding outliers in data
- Data Aggregation
- Reshaping data
- Replace & rename
- and more