```
# merge
# default is to use intersection of keys from both
pd.merge(df1, df2, on="id")
# use only keys from left, may add nans
pd.merge(df1, df2, how="left", on="id")
# how="outer" is for the union of all keys, may add nans
# on can also be a list
pd.concat([df1, df2])  # combine the rows of both tables

# apply
capitalizer = lambda x: x.capitalize()
df["A"] = df["A"].apply(capitalizer)
```

קבצים

```
FILE *fopen(const char *filename, const char *mode);
// returns NULL on failure
int fclose(FILE *fp);
// returns 0 on success or EOF on failure
int fgetc(FILE *fp);
// EOF on end of file or error, otherwise the character
int fputc(int c, FILE *fp);
// EOF on error, otherwise the same character
int fprintf(FILE *fp, const char *format, ...);
// returns the number of characters printed
// fprintf(stdout, ...) is equivalent to printf(...)
int fscanf(FILE *fp, const char *format, ...);
// returns the number of successful inputs
int feof(FILE *fp);
// non-zero when end of file reached
char *fgets(char *s, int size, FILE *fp);
// appends '\0', stops on EOF or newline,
// returns NULL when no characters have been read
int fputs(char *s, FILE *fp);
// without \0, EOF on error, non-negative on success
int fflush(FILE *fp);
// flushes writing, returns 0 on success and EOF on error

int sprintf(char *s, const char *format, ...);
int sscanf(char *s, const char *format, ...);
// like fprintf and scanf but into a string
```

numpy

```
# creating arrays
np.array([1, 2, 3])
np.array([[1, 7]]).shape == (1, 2)
np.array([[1, 7]]).ndim == 2  # dimension
np.array([[1, 7]]).dtype == "int64"
np.array([1, 2], dtype=np.float64)

np.zeros((3, 2, 5))  # all zeros, shape is 3, 2, 5
# that is 3 arrays of 2 arrays of 5 items
np.ones((3, 2, 5))
np.full((3, 2, 5), 7)  # all filled with 7
np.eye(3)              # identity matrix
np.array_equal(np.arange(1, 11, 2),  # like range
               np.array([1, 3, 5, 7, 9]))
np.arange(1, 7).reshape(3, 2)
# [[1,2],[3,4],[5,6]]

# sum
a = np.array([[1, 2, 3], [4, 5, 6]])
np.sum(a, axis=0) == [5, 7, 9]
np.sum(a, axis=1) == [6, 15]
np.sum(a) ==         21  # sum of all items

# slicing
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
a[:2, 1:3]    # [[2, 3], [6, 7]]
a[2, 1:3]     # [10, 11]
a[[0, 2], 0]  # [1, 9]

# operators
x + y, np.add(x, y), x - y, np.subtract(x, y), x * y,
np.multiply(x, y), x / y, np.divide(x, y), np.sqrt(x)

# example of slicing and operators
a = np.array([[1, 2], [3, 4], [5,6]])
b = np.array([1, 0])
c = np.zeros_like(a)  # like a.shape
for i in range(a.shape[0]):
    c[i, :] = a[i, :] + b
# c is [[2, 2], [4, 4], [6, 6]]

# linear algebra
x = np.array([[1, 2], [3, 4]])
v = np.array([9, 10])
x.dot(v), x @ v  # [29, 67]

np.linalg.inv(x)  # inverse matrix
x.T               # transpose
```

pandas

```
names = pd.Series(["A", "B", "C"])
popul = pd.Series([852469, 1015785, 485199])
pd.DataFrame({"City Name":names,"Population":popul})
# if the series length doesn't match the rest is NaN
# you can only check for it with pd.isna, don't do it
df = pd.read_csv("filename.csv")
df["country"] == "Israel"  # lots of booleans
df[(df["country"]=="Israel") & (df["population"]>7000)]
# only relevant rows
df["country"][10]  # country of 11th row

# iloc is by index, loc is by column labels
df.iloc[-1]    # last row
df.iloc[:, 0]  # first column
df.loc[103:107, "country"]
df = df.set_index(0), df = df.set_index("country"),
  df = df.reset_index()  # change the index

# groupby
df.groupby(["Team"]).Salary.mean()
# averages of salaries
# groupby, groupby.Salary returns a DataFrameGroupBy
# and SeriesGroupBy, which are special objects
df.groupby(["Team", "Position"]).Name.count().head(15)
```