C language

```
false=0=NULL, true != 0
#include name                          #define key value
#typedef type name                     #ifdef identifier
%a.bt // a padding, b precision, t type
math.h: sqrt(), pow(), exp(), log(), sin(), cos(), tan() // use double
for (i=0;i<n;i++) = for (i=0;i<n;++i)
auto: local            extern: global            static: private
Arrays are on local stack, pass by value in struct else by reference
a[i][j] = *(*(a + i) + j) = *(&a[0][0] + 2*i + j)
const int* p: *p is const              int* const p: p is const
const int* const p: *p is const and p is const
double f(double (*func)(int)) { return func(1); }
void qsort(void* arr, size_t n, size_t elem, int (*compare)(const
void *, const void *))
```

Types

Type	Size	Format	Pointer	Ends with
char:	1 byte	%c	%s	
short:	2 byte	%hi	%hn	
int:	4 byte	%d	%u	
unsigned:	4 byte	%u	%ls	u
float:	4 byte	%f	%p	f
long:	8 byte	%ld	%ln	l
double:	8 byte	%lf	%p	
long double:	16 byte	%Lf	%p	l
Hexa:	0x10	%x		
Octal:	0o10	%o		

Memory

```
void* malloc(size_t size) // fail: NULL
void* calloc(size_t n, size_t elem) // initialized to zero
void* realloc(void* ptr, size_t new_size) // change size of allocated
memory
void free(void* ptr)
```

Files

```
FILE* fopen(char* filename, char* mode) // fail: NULL
int* fclose(FILE *fp) // success: 0, fail: EOF
int fgetc(FILE* f) // fail: EOF
int fputc(int character, FILE* f) // fail: EOF
int fprintf(FILE* fp, char* formatter, …) // fail: negative
int fscanf(FILE* fp, char* formatter, void* p, …) // fail: EOF
FILE* stdin                    FILE* stdout                 FILE* stderr
size_t fread(void* arr, size_t elem, size_t n, FILE *fp) // read
n*elem bytes (char) from fp and put in arr, fail: 0
size_t fwrite(void* arr, size_t elem, size_t n, FILE *fp) // write
n*elem bytes (char) from arr into fp, fail: 0
#define SEEK_SET 0                     #define SEEK_CUR 1
#define SEEK_END 2
int fseek(FILE* fp, long offset, int place) // change position in fp to
distance offset from place where place is SEEK_X, success: 0
void rewind(FILE* fp) // changes position to start
long ftell(FILE *fp) // get current position, fail: -1
int feof(FILE *fp) // check if got to EOF
char* fgets(char* arr, int length, FILE* fp) // writes into arr length-1
chars (adds '/0' at end), fail: NULL
int fputs(char* str, FILE* fp) // removes '/0' at writing, fail: EOF
int fflush(FILE* fp) // clears fp buffer, success: 0, fail: EOF
```

Strings

```
char* strcat(char* s1, char* s2) // concatenate strings into s1
int strcmp(const char* s1, char* s2) // comparator, return 0 on true
char* strcpy(char* s1, char* s2) // copy s2 into s1
unsigned strlen(char* s) // length of s
int atoi(char* s) // convert s to int, fail: 0
double atof(char* s) // convert s to double, fail: 0.0
char* strdup(char* s) // duplicate s, fail: NULL
char* strstr(char* s1, char* s2) // pointer to s1.find(s2), fail: NULL
char* strchr(char* s, int c) // pointer to s1.find(s2), fail: NULL
most files functions exists for string with s at start instead of f
```

Bitwise

~a	not
a&b	and
a|b	or
a^b	xor
a<<b	left shift
a>>b	right shift

left shift - always pad with zero

right shift - if unsigned pad with zero else with the value of the left

```
limits.h: CHAR_BIT // amount of bits in char (byte)
```

Python language

```
True != False // logical XOR
Bin: 0b10
int.to_bytes(self, length, "big", signed=False)
int.bit_length(self)
str.capitalize(self) // first letter big
str.upper(self) // all big letters
str.rjust(self, width, fillchar=' ') // right pad with fillchar
str.center(self, width, fillcha=' ') // pad with fillchar
str.strip(self, chars) // remove chars in start and end
list.index(self, element) // find element in list
enumerate(iter) // tuples of (index, element)
dict.get(self, key, default=None) // get key if not exist return default
dict.items(self) // tuples of (key, value)
set.add(self, item)
set.union(self, s1)
```

Numpy

```
import numpy as np
np.pi, np.sin(x), np.cos(x), np.log(x) // and more…
np.array(a, copy=True) // a != None
a.shape // tuple of dimensions in a
a.size // number of elements in a
a.ndim // number of dimensions in a
a.T // transpose
a.dtype // data type of a elements
dtype('float64')                              dtype('int64')
np.array_equal(a, b) // returns bool
np.reshape(a, shape) // returns reshaped
np.append(a, values, axis=None)
np.sum(a, axis=None) // if axis in None return type is number,
axis=0 is rows, axis=1 is columns
np.max(a, axis=None)                          np.min(a, axis=None)
np.std(a, axis=None) // standard deviation like sum
np.mean(a, axis=None) // like sum
np.median(a, axis=None) // like sum
np.empty_like(shape) // return empty ndarray of shape
np.zeros(shape) // return ndarray of shape with all 0
np.ones(shape) // return ndarray of shape with all 1
np.full(shape, val) // return ndarray of shape with all val
np.eye(r, c=r, k=0) // matrix of order r on c with diagonal k filled
with 1, k=0 main, k>0 above, k<0 below
np.sort(a, axis=None)
np.arange(start=0, end, jump=1) // range(start, end, jump)
np.linalg.inv(a) // compute matrix reverse of a
np.linalg.eigvals(a) // compute eigenvalues
np.random.seed(0) // random will be the same every run
np.random.rand(d1,…,dN) // return ndarray of shape (d1,…,dN)
with random values in [0, 1)
np.random.normal(m, h, shape) // create ndarray of shape of
normal distribution with middle m and height h
np.linspace(s, e, n) // return n uniform jumps from s to e

two ndarrays are compatible on dimension if they have the same
size or if one of them has size 1.
np.tile(a, shape) // concat a to itself till reached shape
when doing operations between two ndarrays tiling is used to cast
the ndarrays to the same shape (process called broadcasting)
np.add(a, b) // element wise same as: a + b
np.subtract(a, b) // element wise same as: a - b
np.multiply(a, b) // element wise same as: a * b
np.divide(a, b) // element wise same as: a / b
np.sqrt(a) // element wise same as: a**0.5
np.dot(a, b) // dot product same as: a @ b

a[s1:e1:j1, …, sN:eN:jN] // slicing equiv a[s1:e1:j1]…[sN:eN:jN]
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
                a[1, :] // [5, 6, 7, 8]
                a[1:2, :] // [[5, 6, 7, 8]]
                a[[1], :] // [[5, 6, 7, 8]]
                a[[0, 1], [0, 1]] // [1, 6]

Operator between ndarray and number does it element wise.
a = np.array([[1, 2], [3, 4]])
                a > 1 // [[False, True], [True, False]]
                a * 2 // [[2, 4], [6, 8]]
                a[a > 1] // [2, 3, 4]
```

Matplotlib

```
import matplotlib.pyplot as plt
plt.plot(x, y) // x,y arrays
plt.xlabel(s) // lable x axis as s
plt.ylabel(s) // lable y axis as s
plt.title(s) // lable graph as s
plt.legend(a) // add decleration from a to each plot by their order
plt.subplot(row, col, ind) // create row by col subplots and the
current on is ind
plt.show() // show the plot
plt.savefig(s) // save plt with name s
plt.scatter(x, y) // plot dots without lines conecting
plt.figure(figsize=(width, height)) // changes plt size in inches
```

Scipy

```
import scipy
scipy.optimize.minimize_scalar(f).fun // the minimum of f
scipy.stats.norm.pdf(a, m, h) // create ndarray of normal
distribution with middle m and height h
scipy.stats.norm.cdf(a, m, h) // create ndarray of sin curve with
middle m and height h
```

Pandas

```
import pandas as pd
pd.Series(a) // convert array to Series
pd.DataFrame(d, index=None, columns=None) // return
DataSeries, if d is a dict then the keys will be the columns names
and the values are Series, every blank space will be filled with
NaN, index is the names of the rows, if d is an array then columns
is the labels of the columns
DataFrame behaves the same as Numpy arrays with operators
df.shape // shape of DataFrame df
df.to_numpy() // return ndarray
pd.read_csv(s) // s is a filename
df.head(n) // return n first rows of df
df.tail(n) // return n last rows of df
df.describe() // return statistical analysis of df
df.sample(n) // sample n rows randomly from df
df.column // same as df[column]
pd.unique(a) // return ndarray of unique items in a
df.iloc[:, :] // get by numerical index
df.loc[:, s] // get by numerical and label
df.set_index(s) // set the row indices as s labeled column
df[column].value_counts() // how many times does each value in
column appear
df[column].isin(a) // rows where the column value in a
df.drop(labels=None, axis=0) // remove labels from axis
df.drop_duplicates()
df.dropna() // removes all rows with NaN
df.fillna(value=None) // fill all NaN with value
df.groupby(a).mean() // group all same values in the coulmns in a
with mean as group operator
df.groupby(a).sum() // group operator sum
df.sort_values(by=None, ascending=True) // by is a list of the
columns to sort by
pd.merge(df1, df2, left_on=None, right_on=None, how="inner") //
merge df1 and df2 by how method on the left_on columns and the
right_on columns, if there is the same column name in both sides
the left will be called S_x and the right one S_y (were S is the
original name)
pd.concat(dfs) // dfs is an list of DataFrames, concat will append
the DataFrames by rows
df.apply(f) // return a DataFrame with f applied on each element in
df
df.plot(x, y) // same as: plt.plot(df[x], df[y])
```

Faker

```
from faker import Faker
fake = Faker()
fake.name() // fake name
fake.address() // fake address
fake.company() // fake company
```

Sklearn

```
from sklearn import datasets
S = datasets.load_S() // load data named S
S.DESCR // string representing the description of S
S.data // the data itself
S.feature_names // columns names
S.target // the true result of S.data
S.target_names // the names of S.target result
```

Supervised learning – Linear regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x, y) // train the model
model.score(x, y) // between 0,1
model.predict(x) // returns ndarray of the output he thinks good
```

Supervised learning – KNeighbors Classifier

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=per, random_state=0) // split the data into train and test
(0<per<1)
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n) // n is the number of neighbors
model.fit(X_train, y_train) // train the model on the train data
model.predict(X_test) // returns ndarray of the output he thinks
good
np.mean(model.predict(X_test) == y_test) // accuracy score
import sklearn.metrics as merics
metrics.accuracy_score(model.predict(X_test). y_test) // accuracy
metrics.classification_report(model.predict(X_test), y_test) //
DataFrame of the success of the model
from sklearn.model_selection import cross_val_score
cross_val_score(model, X, y) // ndarray of more accuracy tests
```

Unsupervised learning – Kmeans

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
model.labels_ // for each point in X the corresponding cluster
```

Unsupervised learning – Dimensionality reduction

```
from sklearn.preprocessing import StandardScaler
StandardScaler().fit_transform(X) // ndarray of normalized
columns
from sklearn.decomposition import PCA
PCA(n_components=k).fit_transform(X) // reduce number of
columns to k
```

```
promotion in c:
if (long double/double/float/unsigned long and other)
        other to long double/double/float/unsigned long.
else
        integral promotion

integral promotion:
if (long and unsigned)
        if (long can represent all the values of the unsigned)
            unsigned to long.
        else
                both to unsigned long.
elif (long and other)
        other to long.
elif (unsigned and other)
        other to unsigned.
else
        both to int
```