Python Cheat Sheet

List Methods

Method	Purpose
l.append(x)	Adds x to the end of the list.
l.extend(iterable)	Adds all items from iterable to the list.
l.insert(i, x)	Inserts x at index i.
l.index(x, start, end)	Finds the index of x between start and end.
l.remove(x)	Removes the first occurrence of x.
l.sort(key, reverse)	Sorts the list in place. key is a function to transform items before sorting.
l.pop(i)	Removes and returns the item at index i.
l.reverse()	Reverses the list in place.

List Comprehensions

Syntax	Description
[expression for x in iterable]	Creates a list from an iterable.
[x for x in iterable if condition]	Adds a filter condition.

Functional List Operations

Function	Purpose
filter(func, list)	Filters elements using func.
map(func, list)	Applies func to each element.

Strings

Method	Purpose
s.split(sep)	Splits string into a list using sep. Defaults to whitespace.
s.join(iterable)	Joins elements of iterable with s as a separator.
s.strip(chars)	Removes chars from both ends of the string.
s.lower()	Converts string to lowercase.
s.upper()	Converts string to uppercase.

File Handling

- Open: open(filename, mode)
  - Modes: r (read), w (write), a (append).
- Read: file.read(), file.readlines(), file.readline()
- Write: file.write(data)
- Close: file.close()
- Use with open() for automatic cleanup.

FILE *fopen (const char *pszFilename, const char *szMode)

Mode	Description
R	Read (text mode)
w	Write (overwrite, text mode)
a	Append (text mode)
rb	Read (binary mode)
wb	Write (overwrite, binary mode)
ab	Append (binary mode)
r+	Read/Write (text mode)
rb+	Read/Write (binary mode)
wb+	Read/Write (binary mode), create file if needed
ab+	Read/append (binary mode), create file if needed

Reading binary data
fread(&struct_var, sizeof(struct_var), 1, file_ptr);

Writing binary data
fwrite(&struct_var, sizeof(struct_var), 1, file_ptr);

Direct Positioning in a File
int **fseek** (FILE *pFile, long lRelativeByteAddress
, int iSeekMode)

Low level I/O

Mode	Description
open	Opens a file and returns a file descriptor
read	Reads data from a file
write	Writes data to a file
lseek	Sets the position in a file
stat	Gets file metadata
opendir	Opens a directory
readdir	Reads directory entries

```
int read(int fd, void *psbBuf, long lCount)
```

- Beginning with the current file position, **read** reads **`lCount`** bytes into the buffer. The data read can be binary.

```
int write(int fd, void *psbBuf, long lCount)
```

- Beginning with the current file position, **write** writes `lCount` bytes from the buffer to the file.

Inode

- size of the file in bytes
- device ID of the device containing the file
- user ID of the owner
- group ID
- file mode
- timestamps for when the inode was last changed, content last modified, and last access
- link count
- 12 direct pointers to data blocks
- one indirect pointer
- one double indirect pointer
- one triple indirect pointer

Number of data blocks needed (N) = file size / block size

Number of entries per block (E) = block size / length of address

Blocks of direct pointers needed (D) = (N – 12) / E
(round up to integer value)

Blocks of indirect pointers needed (I) = (D – 1) / E
(round up to integer value)

Total blocks needed = N + D + I
(not counting inode)

- pid_t fork(): Creates a child process. Returns:
  - 0: To the child process.
  - Child PID: To the parent process.
  - -1: If failed.

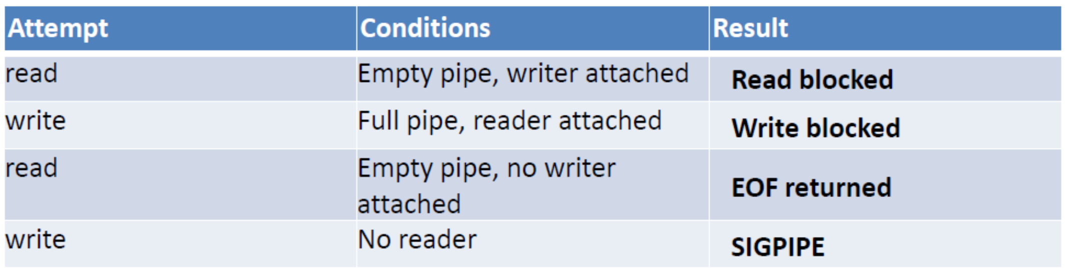
- getpid(): Get current process ID.
- getppid(): Get parent process ID.
- waitpid(): Wait for a child process to finish.
  (avoids creating orphans)

- **Pipe**: one-way communication between parent and child.
- Example: pipe(fd) creates fd[0] for reading and fd[1] for writing.

Named Pipes (FIFOs): Persistent after processes terminate, allowing unrelated processes to communicate.
- Example: mkfifo("mypipe", permissions); open("mypipe", mode)`.

Shared Memory:
  - Fastest IPC method.
  - Requires synchronization (e.g., locks).
  - Steps:
    1. ftok(filename, proj_id): Generate key.
    2. shmget(key, size, flags): Create/get shared memory.
    3. shmat(shmid, NULL, 0): Attach shared memory.
    4. Access and modify shared memory.
    5. shmdt(ptr): Detach memory.
    6. shmctl(shmid, IPC_RMID, NULL): Deallocate memory.

Process States
- **Ready**: Process is waiting to be assigned to CPU.
- **Running**: Process is executing.
- **Waiting**: Waiting for resources (I/O, etc.).
- **Terminated**: Completed execution.



Attempt	Conditions	Result
read	Empty pipe, writer attached	**Read blocked**
write	Full pipe, reader attached	**Write blocked**
read	Empty pipe, no writer attached	**EOF returned**
write	No reader	**SIGPIPE**