Design approach for part 1:

For implementing **PathToInodeNumber(self, path, dir)**, I followed the given pseudo code mentioned in the section 2.5.6 The Path Name Layer. I also implemented three other functions (Plain_name(), first_name(), and rest_name()) which are necessary to implement PathToInodeNumber().

For implementing **GeneralPathToInodeNumber(self, path, cwd)**, I followed the given pseudo code mentioned in the section 2.5.9 The Absolute Path Name Layer.

Design approach for part 2:

To implement **Link(self, target, name, cwd)**, first I find the inode number of the target file by invoking the GeneralPathToInodeNumber() function. Then, I checked three requirements: the target is a file or directory, is there available entry in the inode and whether file name exists in the directory or not. After checking these requirements, I use the InsertFilenameInodeNumber() method and use the filename and the inode number of the target file name as argument.

Design approach for part 3:

mkdir dirname: For implementing mkdir, I followed the Create() method in memoryfs.py and take insight of creating a new directory entry such as allocating data block, inode update etc.

create filename: For implementing create, I followed the Create() method in memoryfs.py and take insight of creating a new file entry.

append filename string: I use Write() method to implement append and use Inode.size as offset.

ln target linkname: I invoked Link() method and used current working directory inode number to implement ln.

ls extension: I just printed the reference count before printing file and directory name.