# CONTEXT MANAGER

**Context Managers** are Python’s resource managers. In most cases, we use files as resources (a simple resource). We often don’t care about closing the files at the end of execution. Context managers are the rescue for this issue by automatically managing resources.

# Python program to demonstrate

# Context Manager

with open('testfile.txt') as in\_file:

    print(''.join(in\_file.readlines()))

IMPLEMENTING context manager:

**# Python program creating a**

**# context manager (File management)**

**class FileManager:**

**def \_\_init\_\_(self, filename):**

**self.filename = filename**

**def \_\_enter\_\_(self):**

**self.opened\_file = open(self.filename)**

**return self.opened\_file**

**def \_\_exit\_\_(self, \*exc):**

**self.opened\_file.close()**

**with FileManager('readme.txt') as file:**

**text = file.read()**

**# Python program shows the**

**# connection management**

**# for MongoDB**

**from pymongo import MongoClient**

**class MongoDBConnectionManager():**

**def \_\_init\_\_(self, hostname, port):**

**self.hostname = hostname**

**self.port = port**

**self.connection = None**

**def \_\_enter\_\_(self):**

**self.connection = MongoClient(self.hostname, self.port)**

**return self**

**def \_\_exit\_\_(self, exc\_type, exc\_value, exc\_traceback):**

**self.connection.close()**

**# connecting with a localhost**

**with MongoDBConnectionManager('localhost', '27017') as mongo:**

**collection = mongo.connection.SampleDb.test**

**data = collection.find({'\_id': 1})**

**print(data.get('name'))**

We can simply make any function as a context manager with the help of **contextlib.contextmanager** decorator without having to write a separate class or \_\_enter\_\_ and \_\_exit\_\_ functions.

**from contextlib import contextmanager**

**@contextmanager**

**def open\_file(filename):**

**opened\_file = open(filename)**

**try:**

**yield opened\_file**

**finally:**

**opened\_file.close()**

All the contents you'd otherwise put in the \_\_enter\_\_ method, except the return statement, goes before the try block here — basically the instructions for opening the resource.

Instead of returning the resource, you yield it, inside a try block.

The contents of the \_\_exit\_\_ method goes inside the corresponding finally block.

**with open\_file('readme.txt') as managed\_file:**

**text = managed\_file.read()**

**print(text)**