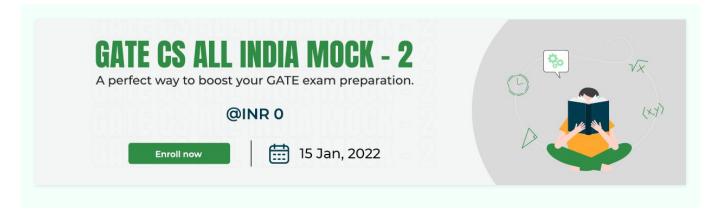
with statement in Python

Difficulty Level: Medium • Last Updated: 15 Feb, 2019

with statement in Python is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams. Observe the following code example on how the use of with statement makes code cleaner.



```
# file handling

# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()

# 2) without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy</u>
Policy

Login

Register

when using with statement. The with statement itself ensures proper acquisition and release of resources. An exception during the file.write() call in the first implementation can prevent the file from closing properly which may introduce several bugs in the code, i.e. many changes in files do not go into effect until the file is properly closed.

The second approach in the above example takes care of all the exceptions but using the with statement makes the code compact and much more readable. Thus, with statement helps avoiding bugs and leaks by ensuring that a resource is properly released when the code using the resource is completely executed. The with statement is popularly used with file streams, as shown above and with Locks, sockets, subprocesses and telnets etc.



Supporting the "with" statement in user defined objects

ere is nothing special in open() which makes it usable with the with statement and the

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy</u>

Login

Register

```
# a simple file writer object

class MessageWriter(object):
    def __init__(self, file_name):
        self.file_name = file_name

def __enter__(self):
        self.file = open(self.file_name, 'w')
        return self.file

def __exit__(self):
        self.file.close()

# using with statement with MessageWriter

with MessageWriter('my_file.txt') as xfile:
        xfile.write('hello world')
```

Let's examine the above code. If you notice, what follows the with keyword is the constructor of MessageWriter. As soon as the execution enters the context of the with statement a MessageWriter object is created and python then calls the __enter__() method. In this __enter__() method, initialize the resource you wish to use in the object. This __enter__() method should always return a descriptor of the acquired resource.

What are resource descriptors?

These are the handles provided by the operating system to access the requested resources. In the following code block, file is a descriptor of the file stream resource.

```
file = open('hello.txt')
```

In the MessageWriter example provided above, the __enter__() method creates a file descriptor and returns it. The name xfile here is used to refer to the file descriptor returned by the __enter__() method. The block of code which uses the acquired

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy</u>
Policy

Login

Register

The contextlib module

A class based context manager as shown above is not the only way to support the with statement in user defined objects. The <u>contextlib</u> module provides a few more abstractions built upon the basic context manager interface. Here is how we can rewrite the context manager for the MessageWriter object using the contextlib module.

```
from contextlib import contextmanager

class MessageWriter(object):
    def __init__(self, filename):
        self.file_name = filename

    @contextmanager
    def open_file(self):
        try:
            file = open(self.file_name, 'w')
            yield file
        finally:
            file.close()

# usage
message_writer = MessageWriter('hello.txt')
with message_writer.open_file() as my_file:
        mv file.write('hello world')
```

Data Structures Algorithms Interview Preparation Topic-wise Practice C++ Java Python C vvnen this open_tile() function is called, it creates a resource descriptor named tile.

This resource descriptor is then passed to the caller and is represented here by the variable my_file. After the code inside the with block is executed the program control

turns hack to the onen file () function The onen file () function resumes its

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy</u>
Policy

Login

Register

requires the knowledge of generators, decorators and yield.

Like 41

Previous

RECOMMENDED ARTICLES

Page: 1 2 3 4 5 6

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy</u>
Policy

Login

Register

15, Nov 19

Q4 Python Continue Statement 20, Nov 19

Article Contributed By:



Vote for difficulty

Current difficulty: Medium

Easy

Normal

Medium

Hard

Expert

Article Tags: Python-exceptions, python-object, Python

Improve Article

Report Issue

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy</u>
Policy

Login

Register

sector-iso, ivolaa, ottar Fradesh - zotsos

feedback@geeksforgeeks.org











Company

About Us

Careers

Privacy Policy

Contact Us

Copyright Policy

Learn

Algorithms

Data Structures

Languages

CS Subjects

Video Tutorials

Web Development

Web Tutorials

HTML

CSS

JavaScript

Bootstrap

Contribute

Write an Article

Write Interview Experience

Internships

Videos

@geeksforgeeks, Some rights reserved

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy