Quick Links for Python
Recent Articles
MCQ / Quizzes
Practice Problems

Basics
Introduction
New Generation Language
Keywords, Set 1 Set 2
Explore More...
Variables
Variables,Expressions & Functions
Global and Local Variables
Type Conversion
Explore More...
Operators
Increment and Decrement Operator
Teranry Operator & Divison Operator
Logical and Bitwise Not Operators on Boolean
Any & ALL

Operator Functions Set 1 & Set 2
Data Types
Introduction
Arrays Set 1, Set 2
String Methods Set 1, Set 2, Set 3
String Template Class & String Formatting using %
List Methods Set 1, Set 2, Set 3
Tuples & Sets
Dictionary Methods Set 1, Set 2
ChainMap
Explore More...
Control Flow
Loops and Control Statements
Counters & Accessing Counters
Iterators & Iterator Functions Set 1, Set 2
Generators
Explore More...
Functions
Function Decorators
Returning Multiple Values
Yield instead of Return
Python Closures & Coroutine
Explore More...
Modules
Introduction

Numeric Functions & Logarithmic and Power functions
Calender Functions Set 1, Set 2
Complex Numbers Introduction & Important functions
Explore More...
Object Oriented Concepts
Class, Object and Members
Data Hiding and Object Printing
Inheritance, Subclass and super
Class method vs static method & Class or Static Variables
Explore More...
Exception Handling
Exception Handling
User-Defined Exceptions
Built-in Exceptions
Libraries and Functions
Timeit
Numpy Set 1, Set 2
Get and Post
import module & reload module
Collection Modules Deque, Namedtuple & Heap
Explore More...
Machine Learning with Python
Classifying data using Support Vector Machines(SVMs) in Python

K means Clustering
How to get synonyms/antonyms from NLTK WordNet in Python?
Explore More...
Misc
Sql using Python & MongoDB and Python
Json formatting & Python Virtual environment
Metaprogramming with Metaclasses in Python
Python Input Methods for Competitive Programming
Explore More...
Applications and Projects
Creating a proxy webserver Set 1, Set 2
Send Messsage to FB friend
Twitter Sentiment Analysis & Whatsapp using Python
Desktop Notifier & Junk File Organizer
Explore More...

File Objects in Python

A file object allows us to use, access and manipulate all the user accessible files. One can read and write any such files.

When a file operation fails for an I/O-related reason, the exception IOError is raised. This includes situations where the operation is not defined for some reason, like seek() on a tty device or writing a file opened for reading.

Files have the following methods:

- **open():** Opens a file in given access mode.

```
open(file_address, access_mode)
```

Examples of accessing a file: A file can be opened with a built-in function called open(). This function takes in the file's address and the access_mode and returns a file object.
There are different types of access_modes:

```
r: Opens a file for reading only
r+: Opens a file for both reading and writing
w: Opens a file for writing only
w+: Open a file for writing and reading.
a: Opens a file for appending
a+: Opens a file for both appending and reading
```

When you add 'b' to the access modes you can read the file in binary format rather than the default text format. It is used when the file to be accessed is not in text.

- **read([size])**: It reads the entire file and returns it contents in the form of a string. Reads at most size bytes from the file (less if the read hits EOF before obtaining size bytes). If the size argument is negative or omitted, read all data until EOF is reached.

```
# Reading a file
f = open(__file__, 'r')

#read()
text = f.read(10)

print(text)
f.close()
```

Run on IDE

- **readline([size])**: It reads the first line of the file i.e till a newline character or an EOF in case of a file having a single line and returns a string. If the size argument is present and non-negative, it is a maximum byte count (including the trailing newline) and an incomplete line may be returned. An empty string is returned only when EOF is encountered immediately.

```
# Reading a line in a file
f = open(__file__, 'r')

#readline()
text = f.readline(20)
print(text)
f.close()
```

Run on IDE

- **readlines([sizehint])**: It reads the entire file line by line and updates each line to a list which is returned.Read until EOF using readline() and return a list containing the lines thus read. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.

```
# Reading a file
f = open(__file__, 'r')
```

```
#readline()
text = f.readlines(25)
print(text)
f.close()
```

Run on IDE

- **write(string)**: It writes the contents of string to the file. It has no return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

```
# Writing a file
f = open(__file__, 'w')
line = 'Welcome Geeks\n'

#write()
f.write(line)
f.close()
```

Run on IDE

More Examples in different modes:

```
# Reading and Writing a file
f = open(__file__, 'r+')
lines = f.read()
f.write(lines)
f.close()
```

Run on IDE

```
# Writing and Reading a file
f = open(__file__, 'w+')
lines = f.read()
f.write(lines)
f.close()
```

Run on IDE

```
# Appending a file
f = open(__file__, 'a')
lines = 'Welcome Geeks\n'
f.write(lines)
f.close()
```

Run on IDE

```
# Appending and reading a file
f = open(__file__, 'a+')
lines = f.read()
f.write(lines)
f.close()
```

Run on IDE

- **writelines(sequence)**: It is a sequence of strings to the file usually a list of strings or any other iterable data type. It has no return value.

```
# Writing a file
f = open(__file__, 'a+')
lines = f.readlines()

#writelines()
f.writelines(lines)
f.close()
```

Run on IDE

- **tell()**: It returns an integer that tells us the file object's position from the beginning of the file in the form of bytes

```
# Telling the file object position
f = open(__file__, 'r')
lines = f.read(10)

#tell()
print(f.tell())
f.close()
```

Run on IDE

- **seek(offset, from_where)**: It is used to change the file object's position. Offset indicates the number of bytes to be moved. from_where indicates from where the bytes are to be moved.

```
# Setting the file object position
f = open(__file__, 'r')
lines = f.read(10)
print(lines)

#seek()
print(f.seek(2,2))
lines = f.read(10)
print(lines)
f.close()
```

Run on IDE

- **flush()**: Flush the internal buffer, like stdio's fflush(). It has no return value. close() automatically flushes the data but if you want to flush the data before closing the file then you can use this method.

```
# Clearing the internal buffer before closing the file
f = open(__file__, 'r')
lines = f.read(10)

#flush()
f.flush()
print(f.read())
f.close()
```

- **fileno()**: Returns the integer file descriptor that is used by the underlying implementation to request I/O

Run on IDE

```
# Getting the integer file descriptor
f = open(__file__, 'r')

#fileno()
print(f.fileno())
f.close()
```

Run on IDE

- **isatty()**: Returns True if the file is connected to a tty(-like) device and False if not.

```
# Checks if file is connected to a tty(-like) device
f = open(__file__, 'r')

#isatty()
print(f.isatty())
f.close()
```

Run on IDE

- **next()**: It is used when a file is used as an iterator. The method is called repeatedly. This method returns the next input line or raises StopIteration at EOF when the file is open for reading(behaviour is undefined when opened for writing).

```
# Iterates over the file
f = open(__file__, 'r')

#next()
try:
    while f.next():
        print(f.next())
except:
    f.close()
```

Run on IDE

- **truncate([size])**: Truncate the file's size. If the optional size argument is present, the file is truncated to (at most) that size. The size defaults to the current position. The current file position is not changed. Note that if a specified size exceeds the file's current size, the result is platform-dependent: possibilities include that the file may remain unchanged, increase to the specified size as if zero-filled, or increase to the specified size with undefined new content.

```
# Truncates the file
f = open(__file__, 'w')

#truncate()
f.truncate(10)
f.close()
```

Run on IDE

- **close()**: Used to close an open file. A closed file cannot be read or written any more.

```
# Opening and closing a file
f = open(__file__, 'r')

#close()
f.close()
```

Run on IDE

- **Attributes**:
  - **closed**: returns a boolean indicating the current state of the file object. It returns true if the file is closed and false when the file is open.
  - **encoding**: The encoding that this file uses. When Unicode strings are written to a file, they will be converted to byte strings using this encoding.
  - **mode**: The I/O mode for the file. If the file was created using the open() built-in function, this will be the value of the mode parameter.
  - **name**: If the file object was created using open(), the name of the file.
  - **newlines**: A file object that has been opened in universal newline mode have this attribute which reflects the newline convention used in the file. The value for this attribute are "\r", "\n", "\r\n", None or a tuple containing all the newline types seen.
  - **softspace**: It is a boolean that indicates whether a space character needs to be printed before another value when using the print statement.

```
f = open(__file__, 'a+')
print(f.closed)
print(f.encoding)
print(f.mode)
print(f.newlines)
print(f.softspace)
```

Run on IDE

Related Article :

Reading and Writing to text files in Python

Reference: https://docs.python.org/2.4/lib/bltin-file-objects.html

This article is contributed by **Sri Sanketh Uppalapati**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Python

Recommended Posts:

Python program to check if a string is palindrome or not
How to input multiple values from user in one line in Python?
How to write an empty function in Python – pass statement?
Reading and Writing to text files in Python
Optimization Tips for Python Code

(Login to Rate and Mark)

0 Average Difficulty : **0/5.0**
No votes yet.

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!