

Input/Output with Python Cheat Sheet

by Nima (nimakarimian) via cheatography.com/113429/cs/23537/

myfile = open("filename.txt", "r")	open and close	reading	w
"w"write mode "a"append mode "r+"write/read mode "rthwrite/read mode "wb"write binary mode "myfile.close() The argument of the open function is the path to the file. Working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Finally: Line 1 text \n', 'Line 2 text \n', 'Line 3 text'] Solution 3 text' You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() Which is determines the number of bytes that we should be read. To retrieve each line in a file, you can use the readlines method file = open("filename.txt", "r") File = open("filename.txt", "r") You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text	<pre>myfile = open("filename.txt")</pre>	<pre>file = open("filename.txt", "r")</pre>	f
"a"append mode "r+"write/read mode "rkb"write binary mode myfile.close() The argument of the open function is the path to the file. working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) print(f.read()) file.close() file = open("filename.txt", "r") print(file.readlines ()) file.close() file = open("filename.txt", "r") print(file.readlines ()) file.close() % ['Line 1 text \n', 'Line 2 text \n', file open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 2 text Line 3 text	"r"read mode	<pre>cont = file.read()</pre>	0
"r+"write/read mode "wb"write binary mode myfile.close() The argument of the open function is the path to the file. working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 2 text Line 2 text Line 3 text	"w"write mode	<pre>print(cont)</pre>	f
"wb"write binary mode myfile.close() The argument of the open function is the path to the file. working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 3 text	"a"append mode	file.close()	W
print(file.read(16)) The argument of the open function is the path to the file. Working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read(6)) This determines the number of bytes that we should be read. d To retrieve each line in a file, you can use the readlines method file = open("filename.txt", "r") print(file.readlines()) file.close() >>> ['Line 1 text \n', 'Line 2 text \n', f' 'Line 3 text'] >>> You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) >> Line 1 text Line 2 text Line 3 text	"r+"write/read mode		f
This determines the number of bytes that w should be read. Working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) To retrieve each line in a file, you can use the readlines method file = open("filename.txt", "r") typint(file.readlines()) with equal to the read in a file, you can use the readlines method file = open("filename.txt", "r") you can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text	"wb"write binary mode	<pre>file = open("filename.txt", "r")</pre>	
myfile.close() The argument of the open function is the path to the file. working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) In this dread. To retrieve each line in a file, you can use the readlines method file = open("filename.txt", "r") the file = open("filename.txt", "bine 2 text \n', file.close() >>>		<pre>print(file.read(16))</pre>	W
The argument of the open function is the path to the file. Working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Convertieve each line in a file, you can use the readlines method file = open("filename.txt", "r") print(file.readlines()) with file = open("filename.txt", 'Line 2 text \n', file 2 text \n', file 2 text \n', for line in file: print(line) statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 3 text		This determines the number of bytes that	W
<pre>working with files It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try; f = open("filename.txt") print(f.read()) finally: f.close()</pre>	<pre>myfile.close()</pre>	should be read.	e
It is good practice to avoid wasting resources by making sure that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) file = open("filename.txt", "r") file.close() >>> ('Line 1 text \n', 'Line 2 text \n', full the distribution of the statement of t	The argument of the open function is the path to the file.		d ·
<pre>that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) print(file.readlines()) file.close() >>> print(file.close() >>> print(file.close() >>> print(file.close() >>> print(file.close() >>> print(file.close() >>> print(file.close() >>> p</pre>	working with files	use the readlines method	T.
that files are always closed after they have been used. try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(file.readlines()) file.close() >>> "Uine 1 text \n', 'Line 2 text \n', "Line 3 text'] >>> ** You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text	It is good practice to avoid wasting resources by making sure	<pre>file = open("filename.txt", "r")</pre>	t:
<pre>try: f = open("filename.txt") print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) file.close() >>> In ['Line 1 text \n', 'Line 2 text \n', f 'Line 3 text'] >>> **You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 2 text Line 3 text</pre>		<pre>print(file.readlines())</pre>	W
<pre>f = open("filename.txt") print(f.read()) finally: f.close()</pre>		file.close()	S
<pre>print(f.read()) finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) ['Line 1 text \n', 'Line 2 text \n', 'Line 3 text'] >>> through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text \n', Line 2 text \n', Line 3 text \n', Line 3 text \n', 'Line 2 text \n', Line 3 text \n', 'Line 2 text \n', 'Line 3 text \n', 'Line 2 text \n', 'Line 3 text \n', 'Line 2 text \n', Line 3 text \n', 'Line 3 text \n', 'Line 2 text \n', Line 3 text \n', 'Line 3 text \n', 'Line 3 text \n', 'Line 2 text \n', 'Line 3 text \n', 'Line 3 text \n', 'Line 3 text \n', 'Line 2 text \n', 'Line 3 tex</pre>	_	>>>	m
<pre>finally: f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) "Line 3 text'] >>> **You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text'] **Nou can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") >> through the lines in the file: file = open("filename.txt", "r") >> through the lines in the file: file = open("filename.txt", "r") >> through the lines in the file: file = open("filename.txt", "r") >> through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt", "r") ** through the lines in the file: file = open("filename.txt"</pre>		['Line 1 text n' , 'Line 2 text n' ,	f
f.close() An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) >>> You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text		'Line 3 text']	t
An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement. The open ("filename.txt") as f: print(f.read()) The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is automatically closed at the end of the with statement. The file is open ("filename.txt", "r") For line in file: print(line) File.close() >>> Line 1 text Line 2 text Line 3 text		>>>	a
An alternative way of doing this is using with statements. This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) You can also use a for loop to iterate through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text			r
This creates a temporary variable (often called f), which is only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) through the lines in the file: file = open("filename.txt", "r") for line in file: print(line) file.close() >>> Line 1 text Line 2 text Line 3 text		You can also use a for loop to iterate	р
<pre>only accessible in the indented block of the with statement. The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 3 text</pre>		through the lines in the file:	f
The file is automatically closed at the end of the with statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 3 text		<pre>file = open("filename.txt", "r")</pre>	>
<pre>statement, even if exceptions occur within it. with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 3 text</pre>		for line in file:	1
<pre>with open("filename.txt") as f: print(f.read()) Line 1 text Line 2 text Line 3 text</pre>		print(line)	>
<pre>print(f.read()) Line 1 text Line 2 text Line 3 text</pre>		file.close()	
Line 1 text Line 2 text Line 3 text		>>>	
Line 3 text	print(1:1eau(//	Line 1 text	
		Line 2 text	
>>>		Line 3 text	
		>>>	



By **Nima** (nimakarimian) cheatography.com/nimakarimian/ www.nimakarimian.ir Published 26th June, 2020. Last updated 26th June, 2020. Page 1 of 1. Sponsored by **Readab**Measure your website
https://readable.com