Python File Iteration

Contents

[Python File Iteration 3](#_Toc91497735)

[>>> python main.py 3](#_Toc91497736)

[Working With Two Files at the Same Time 4](#_Toc91497737)

[Parsing CSV Files With the pandas Library 4](#_Toc91497738)

[Reading CSV Files With **pandas** 5](#_Toc91497739)

[Writing CSV Files With **pandas** 7](#_Toc91497740)

[DataFrames in Pandas 8](#_Toc91497741)

[Sequential Files in Python 8](#_Toc91497742)

[7.1 How to read and write a sequential file in Python 9](#_Toc91497743)

[How To Read CSV to List in Python 9](#_Toc91497744)

[Exporting variable to CSV file in Python 9](#_Toc91497745)

[Pythons write large file in chunks 9](#_Toc91497746)

[Ultimate Guide To Recursion And Iteration In Python 9](#_Toc91497747)

[References 9](#_Toc91497748)

# Python File Iteration

## Python While Loop

|  |  |
| --- | --- |
| **While Loop**   * With the while loop we can execute a set of statements while a condition is true. * remember to increment i, or else the loop will continue forever. | Print i as long as i is less than 6:  i = 1 while i < 6:   print(i)   i += 1 |
| **break Statement:**   * With the break statement we can stop the loop even if the while condition is true: | Exit the loop when i is 3:  i = 1  while i < 6:  print(i)  if i == 3:  break  i += 1 |
| **continue Statement:**   * With the continue statement we can stop the current iteration, and continue with the next iteration: | Continue to the next iteration if i is 3:  i = 0  while i < 6:  i += 1  if i == 3:  continue  print(i)  # Note that number 3 is missing in the result |
| **else Statement:**   * With the else statement we can run a block of code once when the condition no longer is true: | Print a message once the condition is false:  i = 1  while i < 6:  print(i)  i += 1  else:  print("i is no longer less than 6") |

## Python For Loop

|  |  |
| --- | --- |
| **For Loop**   * A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). * With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc. | Print each fruit in a fruit list:  fruits = ["apple", "banana", "cherry"] for x in fruits:   print(x)  The for loop does not require an indexing variable to be set beforehand. |
| **Looping Through a String**   * Strings are iterable objects, they contain a sequence of characters: | Loop through the letters in the word "banana":  for x in "banana":   print(x) |
| **break Statement:**   * With the break statement we can stop the loop before it has looped through all the items: | Exit the loop when x is "banana":  fruits = ["apple", "banana", "cherry"] for x in fruits:   print(x)   if x == "banana":     break  Result is  apple  banana  Exit the loop when x is "banana", but this time the break comes before the print:  fruits = ["apple", "banana", "cherry"] for x in fruits:   if x == "banana":     break   print(x)  Result is  apple |
| **continue Statement**   * With the continue statement we can stop the current iteration of the loop, and continue with the next: | Do not print banana:  fruits = ["apple", "banana", "cherry"] for x in fruits:   if x == "banana":     continue   print(x) |
| **range() Function**   * To loop through a set of code a specified number of times, we can use the range() function * The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number. | Using the range() function:  for x in range(6):   print(x)  The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):  Using the start parameter:  for x in range(2, 6):   print(x)  The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):  Increment the sequence with 3 (default is 1):  for x in range(2, 30, 3):   print(x) |
| **Else in For Loop**   * The else keyword in a for loop specifies a block of code to be executed when the loop is finished: | Print all numbers from 0 to 5, and print a message when the loop has ended:  for x in range(6):   print(x) else:   print("Finally finished!") |
| **pass Statement**   * for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error. | for x in [0, 1, 2]:   pass  # having an empty for loop like this, would raise an error without the pass statement |

## Nested Loops

|  |  |
| --- | --- |
| * A nested loop is a loop inside a loop. * The "inner loop" will be executed one time for each iteration of the "outer loop": | Print each adjective for every fruit:  adj = ["red", "big", "tasty"]  fruits = ["apple", "banana", "cherry"]  for x in adj:  for y in fruits:  print(x, y) |
|  |  |

# Looping and File processing

with open('dog\_breeds.txt', 'r') as reader:

# Read and print the entire file line by line

line = reader.readline()

while line != '': # The EOF char is an empty string

print(line, end='')

line = reader.readline()

# >>> python main.py

Here’s a real world example. In one of my past jobs, I did multiple tests for a hardware device. Each test was written using a Python script with the test script file name used as a title. These scripts would then be executed and could print their status using the \_\_file\_\_ special attribute. Here’s an example folder structure:

project/

|

├── tests/

| ├── test\_commanding.py

| ├── test\_power.py

| ├── test\_wireHousing.py

| └── test\_leds.py

|

└── main.py

Running main.py produces the following:

>>> python main.py

tests/test\_commanding.py Started:

tests/test\_commanding.py Passed!

tests/test\_power.py Started:

tests/test\_power.py Passed!

tests/test\_wireHousing.py Started:

tests/test\_wireHousing.py Failed!

tests/test\_leds.py Started:

tests/test\_leds.py Passed!

I was able to run and get the status of all my tests dynamically through use of the \_\_file\_\_ special attribute.

# Working With Two Files at the Same Time

There are times when you may want to read a file and write to another file at the same time. If you use the example that was shown when you were learning how to write to a file, it can actually be combined into the following:

d\_path = 'dog\_breeds.txt'

d\_r\_path = 'dog\_breeds\_reversed.txt'

with open(d\_path, 'r') as reader, open(d\_r\_path, 'w') as writer:

dog\_breeds = reader.readlines()

writer.writelines(reversed(dog\_breeds))

# [Parsing CSV Files With the pandas Library](https://realpython.com/python-csv/" \l "reading-csv-files-with-csv)

Of course, the Python CSV library isn’t the only game in town. [Reading CSV files](https://realpython.com/pandas-read-write-files/#read-a-csv-file) is possible in [pandas](http://pandas.pydata.org/index.html) as well. It is highly recommended if you have a lot of data to analyze.

pandas is an open-source Python library that provides high performance data analysis tools and easy to use data structures. pandas is available for all Python installations, but it is a key part of the [Anaconda](https://www.anaconda.com/) distribution and works extremely well in [Jupyter notebooks](https://jupyter.org/) to share data, code, analysis results, visualizations, and narrative text.

Installing pandas and its dependencies in Anaconda is easily done:

$ conda install pandas

As is using [pip/pipenv](https://realpython.com/pipenv-guide/) for other Python installations:

$ pip install pandas

We won’t delve into the specifics of how pandas works or how to use it. For an in-depth treatment on using pandas to read and analyze large data sets, check out [Shantnu Tiwari’s](https://realpython.com/team/stiwari/) superb article on [working with large Excel files in pandas](https://realpython.com/working-with-large-excel-files-in-pandas/).

# Reading CSV Files With **pandas**

To show some of the power of pandas CSV capabilities, I’ve created a slightly more complicated file to read, called hrdata.csv. It contains data on company employees:

Name,Hire Date,Salary,Sick Days remaining

Graham Chapman,03/15/14,50000.00,10

John Cleese,06/01/15,65000.00,8

Eric Idle,05/12/14,45000.00,10

Terry Jones,11/01/13,70000.00,3

Terry Gilliam,08/12/14,48000.00,7

Michael Palin,05/23/13,66000.00,8

Reading the CSV into a pandas [DataFrame](https://realpython.com/pandas-dataframe/) is quick and straightforward:

import pandas

df = pandas.read\_csv('hrdata.csv')

print(df)

That’s it: three lines of code, and only one of them is doing the actual work. pandas.read\_csv() opens, analyzes, and reads the CSV file provided, and stores the data in a [DataFrame](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html). Printing the DataFrame results in the following output:

Name Hire Date Salary Sick Days remaining

0 Graham Chapman 03/15/14 50000.0 10

1 John Cleese 06/01/15 65000.0 8

2 Eric Idle 05/12/14 45000.0 10

3 Terry Jones 11/01/13 70000.0 3

4 Terry Gilliam 08/12/14 48000.0 7

5 Michael Palin 05/23/13 66000.0 8

Here are a few points worth noting:

* First, pandas recognized that the first line of the CSV contained column names, and used them automatically. I call this Goodness.
* However, pandas is also using zero-based integer indices in the DataFrame. That’s because we didn’t tell it what our index should be.
* Further, if you look at the data types of our columns , you’ll see pandas has properly converted the Salary and Sick Days remaining columns to numbers, but the Hire Date column is still a String. This is easily confirmed in interactive mode:

>>>

>>> print(type(df['Hire Date'][0]))

<class 'str'>

Let’s tackle these issues one at a time. To use a different column as the DataFrame index, add the index\_col optional parameter:

import pandas

df = pandas.read\_csv('hrdata.csv', index\_col='Name')

print(df)

Now the Name field is our DataFrame index:

Hire Date Salary Sick Days remaining

Name

Graham Chapman 03/15/14 50000.0 10

John Cleese 06/01/15 65000.0 8

Eric Idle 05/12/14 45000.0 10

Terry Jones 11/01/13 70000.0 3

Terry Gilliam 08/12/14 48000.0 7

Michael Palin 05/23/13 66000.0 8

Next, let’s fix the data type of the Hire Date field. You can force pandas to read data as a date with the parse\_dates optional parameter, which is defined as a list of column names to treat as dates:

import pandas

df = pandas.read\_csv('hrdata.csv', index\_col='Name', parse\_dates=['Hire Date'])

print(df)

Notice the difference in the output:

Hire Date Salary Sick Days remaining

Name

Graham Chapman 2014-03-15 50000.0 10

John Cleese 2015-06-01 65000.0 8

Eric Idle 2014-05-12 45000.0 10

Terry Jones 2013-11-01 70000.0 3

Terry Gilliam 2014-08-12 48000.0 7

Michael Palin 2013-05-23 66000.0 8

The date is now formatted properly, which is easily confirmed in interactive mode:

>>>

>>> print(type(df['Hire Date'][0]))

<class 'pandas.\_libs.tslibs.timestamps.Timestamp'>

If your CSV files doesn’t have column names in the first line, you can use the names optional parameter to provide a list of column names. You can also use this if you want to override the column names provided in the first line. In this case, you must also tell pandas.read\_csv() to ignore existing column names using the header=0 optional parameter:

import pandas

df = pandas.read\_csv('hrdata.csv',

index\_col='Employee',

parse\_dates=['Hired'],

header=0,

names=['Employee', 'Hired','Salary', 'Sick Days'])

print(df)

Notice that, since the column names changed, the columns specified in the index\_col and parse\_dates optional parameters must also be changed. This now results in the following output:

Hired Salary Sick Days

Employee

Graham Chapman 2014-03-15 50000.0 10

John Cleese 2015-06-01 65000.0 8

Eric Idle 2014-05-12 45000.0 10

Terry Jones 2013-11-01 70000.0 3

Terry Gilliam 2014-08-12 48000.0 7

Michael Palin 2013-05-23 66000.0 8

[Remove ads](https://realpython.com/account/join/)

# Writing CSV Files With **pandas**

Of course, if you can’t get your data out of pandas again, it doesn’t do you much good. Writing a DataFrame to a CSV file is just as easy as reading one in. Let’s write the data with the new column names to a new CSV file:

import pandas

df = pandas.read\_csv('hrdata.csv',

index\_col='Employee',

parse\_dates=['Hired'],

header=0,

names=['Employee', 'Hired', 'Salary', 'Sick Days'])

df.to\_csv('hrdata\_modified.csv')

The only difference between this code and the reading code above is that the print(df) call was replaced with df.to\_csv(), providing the file name. The new CSV file looks like this:

Employee,Hired,Salary,Sick Days

Graham Chapman,2014-03-15,50000.0,10

John Cleese,2015-06-01,65000.0,8

Eric Idle,2014-05-12,45000.0,10

Terry Jones,2013-11-01,70000.0,3

Terry Gilliam,2014-08-12,48000.0,7

Michael Palin,2013-05-23,66000.0,8

# DataFrames in Pandas

|  |  |
| --- | --- |
| **What is a DataFrame in pandas?**  [[Image result for dataframe in pandas](https://www.google.com/search?q=What+is+a+DataFrame+in+pandas?&rlz=1C1GCEB_enUS969US969&tbm=isch&source=iu&ictx=1&fir=m4QIcNYJ1DmQxM%252CsKSNIenaH6DK_M%252C_&vet=1&usg=AI4_-kRZpJs6_JvTJGxLLcG2w6XRujG23w&sa=X&ved=2ahUKEwjWkemIxI70AhUiSjABHQqeCbkQ9QF6BAgJEAE#imgrc=m4QIcNYJ1DmQxM)](https://www.google.com/search?q=What+is+a+DataFrame+in+pandas?&rlz=1C1GCEB_enUS969US969&tbm=isch&source=iu&ictx=1&fir=m4QIcNYJ1DmQxM%252CsKSNIenaH6DK_M%252C_&vet=1&usg=AI4_-kRZpJs6_JvTJGxLLcG2w6XRujG23w&sa=X&ved=2ahUKEwjWkemIxI70AhUiSjABHQqeCbkQ9QF6BAgJEAE" \l "imgrc=m4QIcNYJ1DmQxM)  Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. ... Indexing and Selecting Data  A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns. | **What is DataFrame used for?**  DataFrame. DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. |
|  |  |

# Sequential Files in Python

<https://people.cs.ksu.edu/~schmidt/200s06/Lectures/7.filesF.html#7.1>

## [7.1 How to read and write a sequential file in Python](https://people.cs.ksu.edu/~schmidt/200s06/Lectures/7.filesF.html" \l "7.1)

*Copyright © 2005 David Schmidt*

<https://people.cs.ksu.edu/~schmidt/200s06/Lectures/7.filesF.html#7.1>

### [How To Read CSV to List in Python](https://www.studytonight.com/python-howtos/how-to-read-csv-to-list-in-python)

### [Exporting variable to CSV file in Python](https://www.geeksforgeeks.org/exporting-variable-to-csv-file-in-python/)

### Pythons write large file in chunks

# [Ultimate Guide To Recursion And Iteration In Python](https://analyticsindiamag.com/ultimate-guide-to-recursion-and-iteration-in-python/)

# References

* [How to import variables from another file in Python?](https://www.geeksforgeeks.org/how-to-import-variables-from-another-file-in-python/)
* [Processing large files using python](https://www.blopig.com/blog/2016/08/processing-large-files-using-python/)
* with statement in Python - <https://www.geeksforgeeks.org/with-statement-in-python/>
* [Parsing CSV Files With the pandas Library](https://realpython.com/python-csv/#reading-csv-files-with-csv) - <https://realpython.com/python-csv/#writing-csv-files-with-pandas>
* [Python Tutorial: Working with CSV file for Data Science](https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/)
* <https://docs.python.org/3/library/csv.html>
* Python Summary - <https://people.cs.ksu.edu/~schmidt/200s06/Lectures/999.summary.html>
* [Exporting variable to CSV file in Python](https://www.geeksforgeeks.org/exporting-variable-to-csv-file-in-python/)
* [Ultimate Guide To Recursion And Iteration In Python](https://analyticsindiamag.com/ultimate-guide-to-recursion-and-iteration-in-python/)