# file-handling

March 30, 2024

[ ]:
```
'''A file is a sequence of bytes and a byte is a number between 0 and 255. That␣
 ↪is, a file is a sequence of integer values.'''
```

[3]:
```
# 1. PathHandling:(https://pythonandvba.com/wpcontent/uploads/2021/11/
 ↪Pathlib_Cheat_Sheet.pdf)
'''We can create Path object to store path of a file for utilization in code.'''
import pathlib
path = pathlib.Path(r"C:\Users\David\Desktop\hello.txt")
path
```

[3]:
```
WindowsPath('C:/Users/David/Desktop/hello.txt')
```

[8]:
```
'''(You can turn the string into a raw string by prefixing it with an r. This␣
 ↪tells Python to ignore any escape sequences and just read the string as is)␣
 ↪Now Path of hello.txt file is store as object path'''
'''Every operating system has a special directory for storing data for the␣
 ↪currently logged in user. This directory is called the user's home directory.
 ↪ The Path.home() class method creates a Path object representing the home␣
 ↪directory:'''
home=pathlib.Path.home()
print(home)
```

```
C:\Users\scimo
```

[9]:
```
'''The pathlib.Path.cwd() class method returns a Path object representing the␣
 ↪current working directory,  or CWD.'''
cwd=pathlib.Path.cwd()
cwd
```

[9]:
```
WindowsPath('C:/Users/scimo')
```

[10]:
```
'''Using the / Operator: If you have an existing Path object, you can use the /␣
 ↪operator to extend the path with subdirectories or file names. For example,␣
 ↪the following creates a Path object representing a file named hello.txt in␣
 ↪the Documents subdirectory of the current user's home directory:'''
path=home / "Desktop" / "hello.txt"
path
```

```
'''The / operator must always have a Path object on the left hand side. The␣
 ↪right hand side can have either string representing a single file or␣
 ↪directory, or a string representing a path, or another Path object.'''
```

[10]:
```
WindowsPath('C:/Users/scimo/Desktop/hello.txt')
```

[11]:
```
'''A path that begins with the root directory in a file system is called an␣
 ↪absolute  path. Not all file paths are absolute. A file path that is not␣
 ↪absolute is called a relative path. You can check whether or not a file path␣
 ↪is absolute using the .is_absolute() method.'''
'''The .parent attribute returns the name of the first parent directory in the␣
 ↪file path as a string: '''
path.parent
```

[11]:
```
WindowsPath('C:/Users/scimo/Desktop')
```

[12]:
```
path.stem
```

[12]:
```
'hello'
```

[13]:
```
path.suffix
```

[13]:
```
'.txt'
```

[14]:
```
'''For instance, if you don't have a hello.txt file in your home directory,␣
 ↪then the .exists() method on the Path object representing that file path␣
 ↪returns False:'''
path.exists()
```

[14]:
```
False
```

[ ]:
```
'''You can check whether or not a file path refers to a file or a directory. To␣
 ↪check if the path is a file, use the .is_file() method and for directory use␣
 ↪.is_dir().
```

[40]:
```
# 2. Creating Directory: To create a new directory, use the Path.mkdir() method.
 ↪ In IDLE's Interactive Window, type the following:
new_dir = pathlib.Path.cwd() / "new_direct"
new_dir.mkdir()
#This will create new directory in CWD
''''If you want to create a new directory if it doesn't exists, but avoid␣
 ↪raising the FileExistsError if it does, then you can set the options␣
 ↪exist_ok parameter of the .mkdir() method to True:'''
new_dir.mkdir(exist_ok=True)
new_dir.exists()
```

[40]:
```
True
```

[41]:
```
# 3. Creating File: You can create the file using the Path.touch() method:
New_file=new_dir/'hello.txt'
New_file.touch()
#This will create file in new dir.
New_file.exists()
```

[41]:
```
True
```

[ ]:
```
# 4. Moving: To move a file or directory, use the .replace() method. Syntax:␣
 ↪source.replace(destination)
'''Where sourceis path object of file/dir to be moved and dest., is  file path␣
 ↪of target folder. If the destination folder already exists, it is completely␣
 ↪replaces with the source folder, which could result in the loss of quite a␣
 ↪bit of data. So we can use:
Syntax: if not destination.exists():
            source.replace(destination)'''
```

[42]:
```
#Deleting File: To delete a file, use the .unlink() method. If you want to␣
 ↪ignore the exception, set the optional missing_ok parameter to True:
New_file.unlink(missing_ok=True)
New_file.exists()
```

[42]:
```
False
```

[43]:
```
# 5. Deleting Directory:To remove a directory, use the .rmdir() method. Keep in␣
 ↪mind that the folder must be empty, otherwise, an OSError exception is␣
 ↪raised:
New_file=new_dir/'hello.txt'
New_file.touch()
#This will create file in new dir
new_dir.rmdir()
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[43], line 5
      3 New_file.touch()
      4 #This will create file in new dir
----> 5 new_dir.rmdir()

File ~\anaconda3\Lib\pathlib.py:1156, in Path.rmdir(self)
   1152 def rmdir(self):
   1153     """
   1154     Remove this directory.  The directory must be empty.
   1155     """
-> 1156     os.rmdir(self)
```

```
OSError: [WinError 145] The directory is not empty: 'C:
  ↪\\Users\\scimo\\new_direct'
```

[44]:
```
new_dir.exists()
```

[44]:
```
True
```

[45]:
```
'''If you need to delete an entire directory, even if it is non-empty, then␣
  ↪pathlib won't help you much. However, you can use the rmtree() function from␣
  ↪the built-in shutil module:'''
import shutil
shutil.rmtree(new_dir)
#Recall that new_dir folders contains file hello.txt, now the folder and all of␣
  ↪it's contents are deleted.
new_dir.exists()
```

[45]:
```
False
```

[48]:
```
# 6. Iteration: Everything in a directory is either a file or a subdirectory.␣
  ↪The Path.iterdir() method returns an iterator over Path objects representing␣
  ↪each item in the directory:
path=pathlib.Path.cwd()
for x in path.iterdir():
    print(x)
```

```
C:\Users\scimo\.anaconda
C:\Users\scimo\.conda
C:\Users\scimo\.condarc
C:\Users\scimo\.continuum
C:\Users\scimo\.ipynb_checkpoints
C:\Users\scimo\.ipython
C:\Users\scimo\.jupyter
C:\Users\scimo\.matplotlib
C:\Users\scimo\.spyder-py3
C:\Users\scimo\3D Objects
C:\Users\scimo\anaconda3
C:\Users\scimo\AppData
C:\Users\scimo\Application Data
C:\Users\scimo\Contacts
C:\Users\scimo\Cookies
C:\Users\scimo\Desktop
C:\Users\scimo\Documents
C:\Users\scimo\Downloads
C:\Users\scimo\Favorites
C:\Users\scimo\File Handling.ipynb
C:\Users\scimo\Fundamental Python.ipynb
C:\Users\scimo\IntelGraphicsProfiles
```

```
C:\Users\scimo\Introductory Pandas.ipynb
C:\Users\scimo\Jedi
C:\Users\scimo\Links
C:\Users\scimo\Local Settings
C:\Users\scimo\Matplotlib Object oriented.ipynb
C:\Users\scimo\mentalmentor
C:\Users\scimo\MicrosoftEdgeBackups
C:\Users\scimo\My Documents
C:\Users\scimo\NetHood
C:\Users\scimo\new_director
C:\Users\scimo\new_directory
C:\Users\scimo\NTUSER.DAT
C:\Users\scimo\ntuser.dat.LOG1
C:\Users\scimo\ntuser.dat.LOG2
C:\Users\scimo\NTUSER.DAT{e7e96f4d-d252-11ee-a52f-a25dab1982cf}.TM.blf
C:\Users\scimo\NTUSER.DAT{e7e96f4d-d252-11ee-a52f-a25dab1982cf}.TMContainer00000
000000000000001.regtrans-ms
C:\Users\scimo\NTUSER.DAT{e7e96f4d-d252-11ee-a52f-a25dab1982cf}.TMContainer00000
000000000000002.regtrans-ms
C:\Users\scimo\ntuser.ini
C:\Users\scimo\Numpy.ipynb
C:\Users\scimo\OneDrive
C:\Users\scimo\Pictures
C:\Users\scimo\PrintHood
C:\Users\scimo\Recent
C:\Users\scimo\Saved Games
C:\Users\scimo\Scipy Equations Solving.ipynb
C:\Users\scimo\Scipy Interpolation.ipynb
C:\Users\scimo\Scipy Mathematical Optimization.ipynb
C:\Users\scimo\Searches
C:\Users\scimo\SendTo
C:\Users\scimo\Start Menu
C:\Users\scimo\Sympy.ipynb
C:\Users\scimo\Templates
C:\Users\scimo\Videos
```

[55]:
```
# For iteration of specific types of files we use glob method which takes␣
 ↪Wildcard character(*,?,etc.) as placeholder of pattern recogonition (see␣
 ↪explanation in attached table):
for x in path.glob('*ipynb'):
    print(x)
```

```
C:\Users\scimo\File Handling.ipynb
C:\Users\scimo\Fundamental Python.ipynb
C:\Users\scimo\Introductory Pandas.ipynb
C:\Users\scimo\Matplotlib Object oriented.ipynb
C:\Users\scimo\Numpy.ipynb
C:\Users\scimo\Scipy Equations Solving.ipynb
```

```
C:\Users\scimo\Scipy Interpolation.ipynb
C:\Users\scimo\Scipy Mathematical Optimization.ipynb
C:\Users\scimo\Sympy.ipynb
```

[64]:
```
#You can convert the return value of .glob() to a list:
list(path.glob('*Scipy*'))
```

[64]:
```
[WindowsPath('C:/Users/scimo/Scipy Equations Solving.ipynb'),
 WindowsPath('C:/Users/scimo/Scipy Interpolation.ipynb'),
 WindowsPath('C:/Users/scimo/Scipy Mathematical Optimization.ipynb')]
```

[ ]:
```
#There is also a shorthand method to doing recursive matching called .rglob()␣
 ↪which matches target files in path folder as well as its subfolders.
```

[71]:
```
# 7. File Handling:
# Opening File:
path = pathlib.Path.home() / "hello.txt"
path.touch()
file = path.open(mode="r")
'''Modes:
"r" for read mode which creates a text file object for reading & raises and␣
 ↪error if the file can't be opened.
"w" for write mode which creates a text file object for writing & owerwrites␣
 ↪all existing data in the file.
"a" for append mode which creates a text file object for appending data to the␣
 ↪end of a file.
Use file.close() method to close file after working.'''
```

[73]:
```
# Writing Data:
file=path.open(mode='w')
file.write('Hello there!')
'''When u set mode="w" in .open(), the contents of the original file are␣
 ↪overwritten. This results in the loss of all of the original data in the␣
 ↪file!
```

[73]:
```
12
```

[75]:
```
#You can append data to the end of a file by opening the file in append mode:
file= path.open(mode="a")
file.write("\nHello")
```

[75]:
```
6
```

[110]:
```
#You can write multiple lines to a file at the same time using the .
 ↪writelines() method.
lines_of_text = ["Hello from Line 1\n","Hello from Line 2\n","Hello from Line␣
 ↪3\n"]
```

```
file=path.open(mode='w')
file.writelines(lines_of_text)
file.close()
```

[111]:
```
# Reading Data:
file= path.open(mode="r")
lines=file.read() # Read full file
print(lines)
```

```
Hello from Line 1
Hello from Line 2
Hello from Line 3
```

[116]:
```
file= path.open(mode="r")
for line in file.readlines():
    print(line)
```

```
Hello from Line 1

Hello from Line 2

Hello from Line 3
```

[123]:
```
#To print Specific line:
file= path.open(mode="r")
for num,line in enumerate(file):
    if num == 2:
        print(line.strip()) #Use strip() to remove leading/trailing whitespace␣
 ↪and newline characters
```

```
Hello from Line 3
```

[125]:
```
#To print Specific element of line:
file= path.open(mode="r")
for num,line in enumerate(file):
    if num == 2:
        print(line[0])
file.close()
```

```
H
```

[126]:
```
#An efficient method of working and closing fie:
'''In Python, you can open and manipulate files using the with statement, which␣
 ↪ensures that the file is properly closed when you're done with it, even if␣
 ↪an exception is raised during file operations. The basic syntax for opening␣
 ↪a file with the with statement is as follows:'''
```

```
with path.open('r') as file:
    content = file.read()
    print(content)
# File is automatically closed when you exit the 'with' block
```

```
Hello from Line 1
Hello from Line 2
Hello from Line 3
```

[170]:
```
# Handling CSV (Comma-seperated Values) Files by CSV Module:
import csv
```

[172]:
```
#csv.writer
temperature_readings = [68, 65, 68, 70, 74, 72]
file_path = pathlib.Path.cwd() / "temperatures.txt"
with file_path.open(mode="w") as file:
    writer = csv.writer(file) #csv.writer() return a CSV writer object with␣
 ↪methods for writing data to the CSV file
    writer.writerow(temperature_readings) #e writer.writerow() method writes a␣
 ↪list to a new row in the CSV file
#This creates a file called temperatures.csv
with file_path.open(mode="r") as file:
    text = file.read()
print(text.strip())
```

```
68,65,68,70,74,72
```

[208]:
```
daily_temperatures = [[68, 65, 68, 70, 74, 72],[67, 67, 70, 72, 72, 70],[68,␣
 ↪70, 74, 76, 74, 73]]
file_path = pathlib.Path.cwd() / "dailytemperatures.csv"
# Write data to the CSV file with consistent line endings
with file_path.open(mode="w", newline='') as file:
# newline='' is ued in open() function when writing to the CSV file. This␣
 ↪ensures that the correct line endings are used regardless of the operating␣
 ↪system, which should prevent extra empty lines when reading the file back in.
    writer = csv.writer(file)
    writer.writerows(daily_temperatures)
# Read and print the content of the CSV file
with file_path.open(mode="r") as file:
    text = file.read()
print(text.strip())
```

```
68,65,68,70,74,72
67,67,70,72,72,70
68,70,74,76,74,73
```

[209]:
```
#CSV files are a great way to store records of sequential data because you can␣
 ↪recover each row of the CSV value as a list:
text.split()
```

[209]:
```
['68,65,68,70,74,72', '67,67,70,72,72,70', '68,70,74,76,74,73']
```

[210]:
```
#csv reader
with file_path.open(mode="r") as file:
    reader = csv.reader(file)
    for row in reader:
        print (row)
```

```
['68', '65', '68', '70', '74', '72']
['67', '67', '70', '72', '72', '70']
['68', '70', '74', '76', '74', '73']
```

[211]:
```
#Reading and Writing CSV Files With Headers (Attributes in first row):
header=['t1','t2','t3','t4','t5','t6']
with file_path.open(mode="r") as file:
    reader = csv.reader(file)
    rows = list(reader)
    rows.insert(0, header)  # Insert the header at index 0 (beginning)
with file_path.open(mode="w",newline='') as file:
    writer = csv.writer(file)
    writer.writerows(rows) # Write the updated content back to the file
#Let's read updated file:
with file_path.open(mode="r") as file:
    reader = csv.reader(file)
    for row in reader:
        print (row)
```

```
['t1', 't2', 't3', 't4', 't5', 't6']
['68', '65', '68', '70', '74', '72']
['67', '67', '70', '72', '72', '70']
['68', '70', '74', '76', '74', '73']
```

[236]:
```
#Reading row elements as dictionary based on header:
with file_path.open(mode="r") as file:
    reader = csv.DictReader(file) #When you create a DictReader object, the␣
 ↪first row of the CSV file is assumed to contain the field names. These␣
 ↪values get stored in a list and assigned to the DictReader instance's .
 ↪fieldnames attribute:
    header=reader.fieldnames
    for row in reader:
        print(row)
print(header)
```

```
{'t1': '68', 't2': '65', 't3': '68', 't4': '70', 't5': '74', 't6': '72'}
```

```
{'t1': '67', 't2': '67', 't3': '70', 't4': '72', 't5': '72', 't6': '70'}
{'t1': '68', 't2': '70', 't3': '74', 't4': '76', 't5': '74', 't6': '73'}
['t1', 't2', 't3', 't4', 't5', 't6']
```

[244]:
```
#Converting List of Dictionaries to CSV:
people = [{"name": "Veronica", "age": 29}, {"name": "Audrey", "age":␣
 ↪32},{"name": "Sam", "age": 24}]
file_path = pathlib.Path.cwd() / "people.csv"
with file_path.open(mode="w", newline='') as file:
    writer = csv.DictWriter(file, fieldnames=["name", "age"])
    writer.writerows(people)
with file_path.open(mode="r", newline='') as file:
    reader=csv.reader(file)
    for rows in reader:
        print(rows)
```

```
['Veronica', '29']
['Audrey', '32']
['Sam', '24']
```

| Wildcard Character | Description | Example | Matches | Does Not Match |
|---|---|---|---|---|
| * | Any number of characters | "*b*" | b, ab, bc, abc | a, c, ac |
| ? | A single character | "?bc" | abc, bbc, cbc | bc, aabc, abcd |
| [abc] | Matches one character in the brackets | [CB]at | Cat, Bat | at, cat, bat |