File and exceptions

July 31, 2022

1 Files and exceptions

While a program is running, its data is in memory. When the program ends, or the computer shuts down, data in memory disappears. To store data permanently, you have to put it in a file. Files are usually stored on a hard drive, floppy drive, or CD-ROM. When there are a large number of files, they are often organized into directories (also called "folders"). Each file is identified by a unique name, or a combination of a file name and a directory name. By reading and writing files, programs can exchange information with each other and generate printable formats like PDF. Working with files is a lot like working with books. To use a book, you have to open it. When you're done, you have to close it. While the book is open, you can either write in it or read from it. In either case, you know where you are in the book. Most of the time, you read the whole book in its natural order, but you can also skip around. All of this applies to files as well. To open a file, you specify its name and indicate whether you want to read or write.

[1]:
```
# Opening a file creates a file object. In this example, the variable f refers␣
↪to the new file object.
file = open("abc.txt", "w")
print(file)
```

```
<_io.TextIOWrapper name='abc.txt' mode='w' encoding='cp1252'>
```

The open function takes two arguments. The first is the name of the file, and the second is the mode. Mode "w" means that we are opening the file for writing. If there is no file named test.dat, it will be created. If there already is one, it will be replaced by the file we are writing. When we print the file object, we see the name of the file, the mode, and the location of the object.

[2]:
```
# To put data in the file we invoke the write method on the file object:
file.write("This is first line\n")
file.write("We are writing a file using python")
```

[2]:
```
34
```

[3]:
```
# Closing the file tells the system that we are done writing and makes the file␣
↪available for reading:
file.close()
```

Now we can open the file again, this time for reading, and read the contents into a string. This time, the mode argument is "r" for reading:

[4]:
```
file = open("abc.txt", "r")
```

[5]:
```
# If we try to open a file that doesn't exist, we get an error:
file = open("test.docx","r")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [5], in <cell line: 2>()
      1 # If we try to open a file that doesn't exist, we get an error:
----> 2 file = open("test.docx","r")

FileNotFoundError: [Errno 2] No such file or directory: 'test.docx'
```

[6]:
```
# Not surprisingly, the read method reads data from the file. With no␣
 ↪arguments, it reads the entire contents of the file:

file = open("abc.txt", "r")
data = file.read()
print(data)
```

```
This is first line
We are writing a file using python
```

[7]:
```
# read can also take an argument that indicates how many characters to read:

file = open("abc.txt", "r")
data = file.read(9)
print(data)
```

```
This is f
```

A text file is a file that contains printable characters and whitespace, organized into lines separated by newline characters. Since Python is specifically designed to process text files, it provides methods that make the job easy.

[8]:
```
# To demonstrate, we'll create a text file with three lines of text separated␣
 ↪by newlines:

file = open("abc.txt", "w")
file.write("line one\nline two\nline three\n")
file.close()

# The readline method reads all the characters up to and including the next␣
 ↪newline character:

file = open("abc.txt", "r")
print(file.readline())
```

```
print(file.readline())
print(file.readline())
print(file.readline())     #empty string
print(file.readlines())    #empty list
```

```
line one

line two

line three


[]
```

Exceptions

Whenever a runtime error occurs, it creates an exception. Usually, the program stops and Python prints an error message. For example, dividing by zero creates an exception:

print(55/0)

ZeroDivisionError: integer division or modulo

[9]:
```
# So does accessing a nonexistent list item:
a = []
print(a[4])
# IndexError: list index out of range
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [9], in <cell line: 3>()
      1 # So does accessing a nonexistent list item:
      2 a = []
----> 3 print(a[4])

IndexError: list index out of range
```

[10]:
```
# Or accessing a key that isn't in the dictionary:
b = {}
print(b['what'])
# KeyError: what
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      1 # Or accessing a key that isn't in the dictionary:
      2 b = {}
----> 3 print(b['what'])
```

```
KeyError: 'what'
```

[11]:
```
# Or trying to open a nonexistent file:

file = open("Idontexist", "r")
# IOError: [Errno 2] No such file or directory: 'Idontexist'
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [11], in <cell line: 3>()
      1 # Or trying to open a nonexistent file:
----> 3 file = open("Idontexist", "r")

FileNotFoundError: [Errno 2] No such file or directory: 'Idontexist'
```

In each case, the error message has two parts: the type of error before the colon, and specifics about the error after the colon. Normally Python also prints a traceback of where the program was, but we have omitted that from the examples. Sometimes we want to execute an operation that could cause an exception, but we don't want the program to stop. We can handle the exception using the try and except statements.

[12]:
```
# For example, we might prompt the user for the name of a file and then try to␣
 ↪open it.
# If the file doesn't exist, we don't want the program to crash; we want to␣
 ↪handle the exception:

filename = input('Enter a file name: ')
try:
  file = open (filename, "r")
except IOError:
  print('There is no file named', filename)

# The try statement executes the statements in the first block. If no␣
 ↪exceptions occur, it ignores the except statement.
# If an exception of type IOError occurs, it executes the statements in the␣
 ↪except branch and then continues.
```

```
Enter a file name: xyz.xlsx
There is no file named xyz.xlsx
```

[13]:
```
# If your program detects an error condition, you can make it raise an␣
 ↪exception.
# Here is an example that gets input from the user and checks for the value 17.
# Assuming that 17 is not valid input for some reason, we raise an exception.
```

```
x = float(input('Pick a number: '))
if x == 17:
    raise ValueError('17 is a bad number')
else:
    print("Your number is Okay.")
```

```
Pick a number: 15
Your number is Okay.
```

CSV File Reading and Writing

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, "write this data in the format preferred by Excel," or "read data from this file which was generated by Excel," without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

[14]:
```
# importing the csv module
import csv

filename = "iris.csv"

# initialize Columns
fields = []

# initialize Rows
rows = []

# reading csv file
with open(filename, "r") as csv_file:

#   creating a csv reader object
    csv_reader = csv.reader(csv_file)

#   extracting field names through first row
    fields = next(csv_reader)

#   extracting each data row one by one
    for row in csv_reader:
        rows.append(row)

#   get total number of rows
    print("Total number of rows are {}".format(csv_reader.line_num))

###################
# print all data #
###################
```

```
# printing the field names
print("Field names are: " + ", ".join(field for field in fields))

# first ten rows
print("\nHere are first 10 rows from given data")

for row in rows[:10]:
#   parsing each column of a row
    for col in row:
        print(col + "\t", end = '')
    print("\n")
```

```
Total number of rows are 40
Field names are: Height, number, setosa, versicolor

Here are first 10 rows from given data
5.1     3.5     1.4     0.2

4.9     3       1.4     0.2

4.7     3.2     1.3     0.2

4.6     3.1     1.5     0.2

5       3.6     1.4     0.2

5.4     3.9     1.7     0.4

4.6     3.4     1.4     0.3

5       3.4     1.5     0.2

4.4     2.9     1.4     0.2

4.9     3.1     1.5     0.1
```

2 Writing to a CSV file

[15]:
```
# importing the csv module
import csv

# Columns
fields = ["ID", "Name", "Dept", "CGPA", "Batch"]

# Rows
```

```
rows = [
    ['2001501', 'Ali', 'CS', '3.1', '2020'],
    ['1901502', 'Aeman', 'CS', '2.7', '2019'],
    ['1801503', 'Uzba', 'SE', '3.9', '2018'],
    ['2101504', 'Uzair', 'CS', '4.0', '2021'],
    ['1701505', 'Wahab', 'SE', '2.4', '2017']
]

# File name
filename = "records.csv"

# writing to csv
with open(filename, "w") as csv_file:

#   creating a csv writer object
    csv_writer = csv.writer(csv_file)

#   writing the fields
    csv_writer.writerow(fields)

#   writing the rows
    csv_writer.writerows(rows)

########################
# let's open our file #
########################
```

[16]:
```
# Import Datasets into Python using Pandas

# import module
import pandas as mypanda

# file name with r mode
data = mypanda.read_csv(r"person.csv")

print(data)
```

```
     Name Couuntry    Product    Price
0     Jon    Japan    Cmputer    $800
1    Bill       US     Tablet    $450
2   Maria   Canada    Printer    $150
3    Rita   Brazil     Laptop  $1,200
4    Jack       UK    Monitor    $300
5    Jeff    China  Projector    $500
6  Carrie    Italy        Mic    $100
7     Ben   Russia     Camera  $1,500
8   Chips  Ukarine    Printer    $150
```

[17]:
```
# import module
import pandas as mypanda

# file name with r mode
data = mypanda.read_csv(r"person.csv")

# data frame to filter data
specific = mypanda.DataFrame(data, columns=['Name', 'Product'])

print(specific)
```

```
     Name    Product
0     Jon    Cmputer
1    Bill     Tablet
2   Maria    Printer
3    Rita     Laptop
4    Jack    Monitor
5    Jeff  Projector
6  Carrie        Mic
7     Ben     Camera
8   Chips    Printer
```

[ ]:
```

```