Files in Python

What You Need In Order To Read Information From A File

1. Open the file and associate the file with a file variable.
2. A command to read the information.
3. A command to close the file.

1. Opening Files

Prepares the file for reading:

A. Links the file variable with the physical file (references to the file variable are references to the physical file).
B. Positions the file pointer at the start of the file.

Format:[1]

```
<file variable> = open(<file name>, "r")
```

Example:

(Constant file name)

```
inputFile = open("data.txt", "r")
```

OR

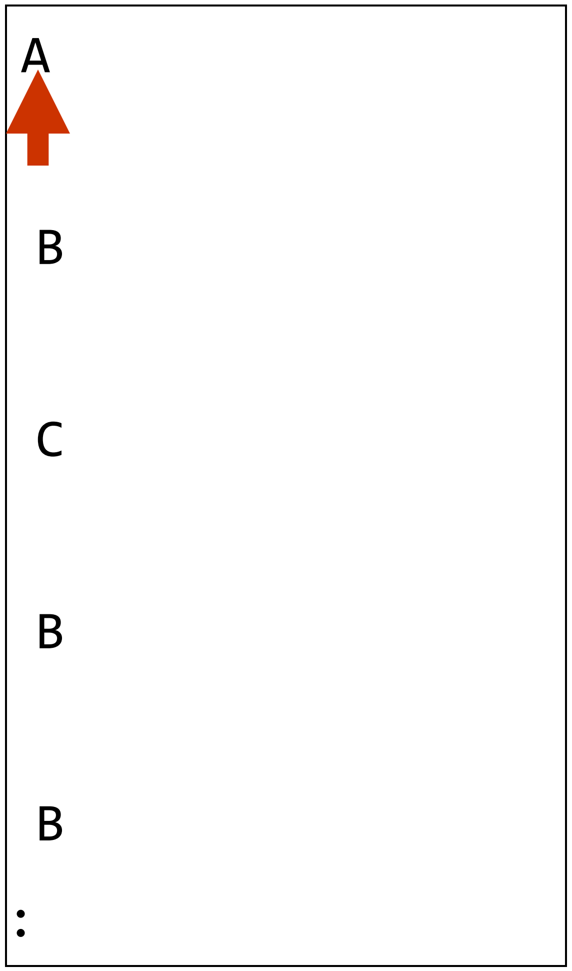
(Variable file name: entered by user at runtime)

```
filename = input("Enter name of input file: ")
inputFile = open(filename, "r")
```

1 Examples assume that the file is in the same directory/folder as the Python program.

B. Positioning The File Pointer

`letters.txt`



2. Reading Information From Files

- Typically reading is done within the body of a loop
- Each execution of the loop will read a line from the file into a string

Format:

```
for <variable to store a string> in <name of file variable>:
    <Do something with the string read from file>
```

Example:

```
for line in inputFile:
    print(line)  # Echo file contents back onscreen
```

Closing The File

- Although a file is automatically closed when your program ends it is still a good style to explicitly close your file as soon as the program is done with it.
  - What if the program encounters a runtime error and crashes before it reaches the end? The input file may remain 'locked' an inaccessible state because it's still open.
- **Format:**

```
<name of file variable>.close()
```

- **Example:**

```
inputFile.close()
```

Reading From Files: Putting It All Together

Name of the online example: `grades1.py`

Input files: `letters.txt or gpa.txt`

```
inputFileName = input("Enter name of input file: ")
inputFile = open(inputFileName, "r")
print("Opening file", inputFileName, " for reading.")

for line in inputFile:
    sys.stdout.write(line)


inputFile.close()
print("Completed reading of file", inputFileName)
```

What You Need To Write Information To A File

1. Open the file and associate the file with a file variable (file is “locked” for writing).
2. A command to write the information.
3. A command to close the file.

1. Opening The File

Format[1]:

```
<name of file variable> = open(<file name>, "w")
```

Example:

(Constant file name)

```
outputFile = open("gpa.txt", "w")
```

(Variable file name: entered by user at runtime)

```
outputFileName = input("Enter the name of the output file
                        to record the GPA's to: ")
outputFile = open(outputFileName, "w")
```

1 Typically the file is created in the same directory/folder as the Python program.

3. Writing To A File

- You can use the '`write()`' function in conjunction with a file variable.
- Note however that this function will ONLY take a string parameter (everything else must be converted to this type first).

Format:

```
outputFile.write(temp)
```

Example:

```
# Assume that temp contains a string of characters.
outputFile.write (temp)
```

Writing To A File: Putting It All Together

- Name of the online example: `grades2.py`
- Input file: "`letters.txt`" (sample output file name: `gpa.txt`)

```
inputFileName = input("Enter the name of input file to read the
                       grades from: ")
outputFileName = input("Enter the name of the output file to
                       record the GPA's to: ")

inputFile = open(inputFileName, "r")
outputFile = open(outputFileName, "w")

print("Opening file", inputFileName, " for reading.")
print("Opening file", outputFileName, " for writing.")
gpa = 0
```

Writing To A File: Putting It All Together (2)

```
for line in inputFile:
    if (line[0] == "A"):
        gpa = 4
    elif (line[0] == "B"):
        gpa = 3
    elif (line[0] == "C"):
        gpa = 2
    elif (line[0] == "D"):
        gpa = 1
    elif (line[0] == "F"):
        gpa = 0
    else:
        gpa = -1
    temp = str (gpa)
    temp = temp + '\n'
    print (line[0], '\t', gpa)
    outputFile.write (temp)
```

Writing To A File: Putting It All Together (3)

```
inputFile.close ()
outputFile.close ()
print ("Completed reading of file", inputFileName)
print ("Completed writing to file", outputFileName)
```

Reading From Files: Commonly Used Algorithm

- Pseudo-code:

```
Read a line from a file as a string
While (string is not empty)
    process the line
    Read another line from the file
```

File Input: Alternate Implementation

- Name of the online example: `grades3.py`

```
inputFileName = input ("Enter name of input file: ")
inputFile = open(inputFileName, "r")
print("Opening file", inputFileName, " for reading.")

line = inputFile.readline()

while (line != ""):
    sys.stdout.write(line)
    line = inputFile.readline()

inputFile.close()
print("Completed reading of file", inputFileName)
```

Data Processing: Files

- Files can be used to store complex data given that there exists a predefined format.
- Format of the example input file: ‘`employees.txt`’

  <Last name><SP><First Name>,<Occupation>,<Income>

Example Program: `data_processing.py`

```
# EMPLOYEES.TXT
Adama Lee,CAG,30000
Morris Heather,Heroine,0
Lee Bruce,JKD master,100000
```

```
inputFile = open ("employees.txt", "r")

print ("Reading from file input.txt")
for line in inputFile:
    name,job,income = line.split(',')
    last,first = name.split()
    income = int(income)
    income = income + (income * BONUS)
    print("Name: %s, %s\t\t\tJob: %s\t\tIncome $%.2f"
             %(first,last,job,income))

print ("Completed reading of file input.txt")
inputFile.close()
```

Error Handling With Exceptions

- Exceptions are used to deal with extraordinary errors (‘exceptional ones’).
- Typically these are fatal runtime errors (“crashes” program)
- Example: trying to open a non-existent file
- Basic structure of handling exceptions

```
try:
    Attempt something where exception error may happen
except <exception type>:
    React to the error
else:  # Not always needed
    What to do if no error is encountered
finally:  # Not always needed
    Actions that must always be performed
```

Exceptions: File Example

- Name of the online example: `file_exception.py`
- Input file name: Most of the previous input files can be used e.g. “`input1.txt`”

```
inputFileOK = False
while (inputFileOK == False):
   try:
      inputFileName = input("Enter name of input file: ")
      inputFile = open(inputFileName, "r")
   except IOError:
      print("File", inputFileName, "could not be opened")
   else:
      print("Opening file", inputFileName, " for reading.")
      inputFileOK = True

      for line in inputFile:
         sys.stdout.write(line)
      print ("Completed reading of file", inputFileName)
      inputFile.close()
      print ("Closed file", inputFileName)
```

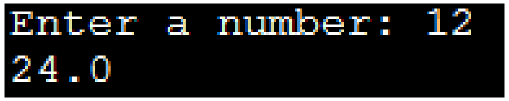
Exceptions: File Example (2)

```
# Still inside the body of the while loop (continued)
finally:
   if (inputFileOK == True):
      print ("Successfully read information from file",
              inputFileName)
   else:
      print ("Unsuccessfully attempted to read information
              from file", inputFileName)
```

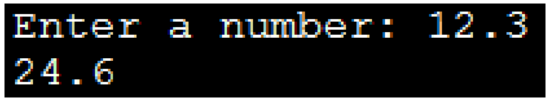
Exception Handling: Keyboard Input

- Name of the online example: `exception_validation.py`

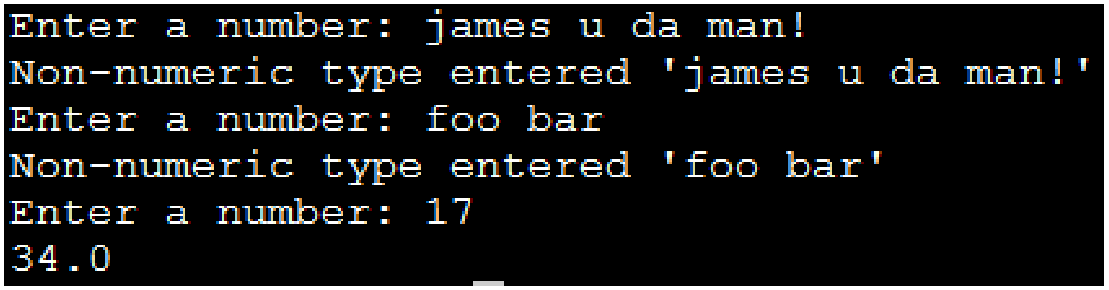
```
inputOK = False
while (inputOK == False):
    try:
        num = input("Enter a number: ")
        num = float(num)
    except ValueError:    # Can’t convert to a number
        print("Non-numeric type entered '%s'" %num)
    else:   # All characters are part of a number
        inputOK = True
num = num * 2
print(num)
```



```
Enter a number: 12
24.0
```



```
Enter a number: 12.3
24.6
```



```
Enter a number: james u da man!
Non-numeric type entered 'james u da man!'
Enter a number: foo bar
Non-numeric type entered 'foo bar'
Enter a number: 17
34.0
```

You Should Now Know

- How to open a file for reading
- How to open a file a file for writing
- The details of how information is read from and written to a file
- How to close a file and why it is good practice to do this explicitly
- How to read from a file of arbitrary size
- Data storage and processing using files and string functions
- How exceptions can be used in conjunction with file input and with invalid keyboard/console input