Introduction To Files In Python

In this section of notes you will learn how to read from and write to text files as well as how to design programs that can recover from runtime errors. To properly read dynamic file information, building variable sized 2D lists is introduced.

What You Need In Order To Read Information From A File

1. Open the file and associate the file with a file variable (the latter positions the “file pointer”.
2. A command to read the information.
3. A command to close the file.

1. Opening Files

- Prepares the file for reading:
  - As the file is opened, there's a link between the file variable and the physical file (references to the file variable are references to the physical file).
  - 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")
```

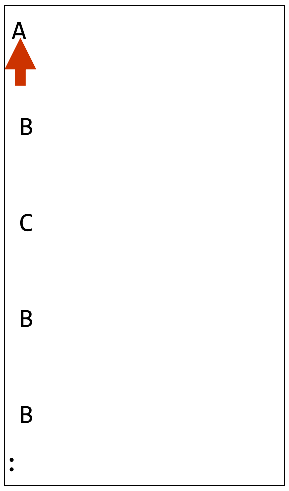
Modes when opening a file
- "r" open file for reading
- "w" open file for writing
- "c" open file for reading or writing, if the file doesn't exist then create it
- "n" create a new file for reading or writing, if the file exists then it's contents are overwritten.
- "a" open the file for appending, create the file if it doesn't exist

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

James Tam

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 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
```

3. 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 example program: `1grades.py`

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

– Learning: reading text information from a file on a line by line basis.

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

for line in inputFile:
    print(line, end="")

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, the file pointer is positioned in the file).
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’s argument:
  - It will ONLY take a string parameter
  - Everything else must be converted to this type first via the ‘`str()`’ function.
- Unlike the `print()` function the `write()` function:
  - Writes to the output file exactly as specified
  - (No extra spaces or newlines are added)

Format:

```
outputFile.write(<string to write to file>)
```

Example:

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

Writing To A File: Putting It All Together

- **Name of the example program**: `2grades.py`
- **Input file**: `letters.txt` (sample output file name: `gpa.txt`)
  - Learning: processing data and writing a line at a time to a file.

```
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 + ENTER
    print(line[0], TAB, 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 (If There Is Time)

- Pseudo-code:

```
Read a line from a file as a string
While (string is not empty)
    process the line e.g. display onscreen, use data in some
     calculations etc.
    Read another line from the file
```

James Tam

File Input: Alternate Implementation

- **Name of the example program**: `3grades.py`
- **Input**: Any of the '.txt' files supplied with this section.
  - Learning: reading from a file using a general approach (not specific to Python but can be applied to other languages).

```
EMPTY = ""
inputFileName = input("Enter name of input file: ")
inputFile = open(inputFileName, "r")
print("Opening file", inputFileName, " for reading.")
line = inputFile.readline()
while (line != EMPTY):
    print(line, end="")
    line = inputFile.readline()

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

James Tam

Data Processing: Files

- Data processing from (https://www.britannica.com)
  – "Manipulation of data by a computer."
  – (Paraphrasing the rest of the definition: converting or processing data from a machine-stored form to a form that is usable).
- Files can be used to store complex data given there exists a predefined format.
- Format of the example input file: '`employees.txt`'
  <Last name><SP>*<First Name>*,*<Occupation>*,*<Income>*

James Tam

Name Of Example Program: `4data_processing.py`

Input file: `Employees.txt`

Learning: After reading information from a file applying text processing in order make sense or make use of the information.

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

```
BONUS = 0.15
inputFile = open("employees.txt", "r")
print("Reading from file input.txt")
for line in inputFile:
    name,job,income = line.split(",") #Fields divided by a comma
    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()
```

James Tam

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:
    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 example program**: `5file_exception.py`
- Input file: Most of the previous input files can be used e.g. “input1.txt”
  - Defining a program to allow it to recover and continue execution if file input/output problems occur.

```
inputFileOK = False
while (inputFileOK == False):
    try:
        inputFileName = input("Enter name of input file: ")
        inputFile = open(inputFileName, "r")
        print("Opening file" + inputFileName, “)
        for line in inputFile:
            print(line, end="")
        print("Completed reading of file", inputFileName)
        inputFileOK = True
```

James Tam

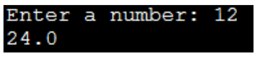
Exceptions: File Example (2)

```
# All this is inside the body of the while loop (continued)
        inputFile.close()
        print("Closed file", inputFileName) # End of try-body
    except IOError:
        print("Error: File", inputFileName, "could not be "+ \
              "opened")
    else:
        print("Successfully read information from file", \
              inputFileName)    finally:
        print("Finished file input and output")
```

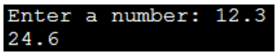
Exception Handling: Keyboard Input

- **Name of the example program**: `6exception_validation.py`
  - Learning: writing a program that can check for and recover when an invalid type has been entered.

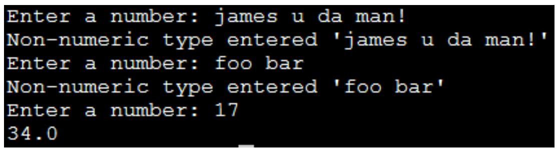
```
inputOK = False
while (inputOK == False):
    try:
        num = input("Enter a number: ")
        num = float(num)
    except ValueError:    # Can't convert to a number
        print("Non-numeric type entered '%s'" %num)
    else:   # All characters are part of a number
        inputOK = True
num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

Creating 2D Lists: What You Know

- **Method 1**: Both dimensions are hard-coded (typed in manually).
  - Drawback: fixed size during creation.

```
matrix = [ [0, 0, 0],
           [1, 1, 1],
           [2, 2, 2],
           [3, 3, 3]]
```

- **Method 2 (may just be covered in tutorial)**: Both dimensions can vary in size.
  - Each element is the same when the list is created.
  - May be tricky if the number of rows and columns not known in advance (e.g. reading data from a file when the size is unknown in advance).

```
aList = []
for r in range(0,MAX_ROWS,1):
    tempRow = ["*"] *
MAX_COLUMNS
    aList.append(tempRow)
```

James Tam

Creating 2D Lists: A New Approach

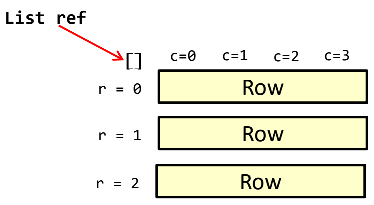
- **Method 3**: the rows and columns can be dynamically created but each element need not be identical.
  - Create a reference to an empty list.
  - Create a new row (reference to a 1D list) and append the new row to the list.
  - Successively append new elements (can be different values) onto the end of the 1D list until the desired number of elements has been added to the row.
  - Repeat the creation of new rows until the desired number of rows has been created.

James Tam

Creating And Initializing A Multi-Dimensional List In Python: Dynamic Creation

General structure (Using loops):

- Create a variable that refers to an empty list
- Create list:
  - One loop (outer loop) traverses the rows.
  - Each iteration of the outer loop creates a new 1D list (empty at start)
  - Then the inner loop traverses the columns of the newly created 1D list creating and initializing each element in a fashion similar to how a single 1D list was created and initialized (add to end)
- Repeat the process for each row in the list



Etc.

```
aGrid = []
for r in range (0, 3, 1):
    aGrid.append ([])
    for c in range (0, 3, 1):
        aValue = <Some source>
        aGrid[r].append(aValue)
```

Repeating Just The Steps In The Code Creating The List

1. Create a variable that refers to an empty list

```
aGrid = []
```

2. Successively create rows in the list

```
for r in range (0,noRows,1):
    aGrid.append ([])
```

3. Each row is a 1D list, add elements to the end of the 1D list (empty list needed in #2 so that the append method can be called to add elements to the end).

```
    for c in range (0,noColumns,1):
        aGrid[r].append("*")
```

– The [r] part of specifies which row the loop will add elements on the end.
`aGrid[r].append("*")`

Recall 'append' is unique to a list. This won't work but an empty row can have new elements appended.

```
num = 123
num.append(4)
```

James Tam

Example 2D List Program: A Variable Sized 2D List (Dynamic)

- **Name of the example program:** `7variable2DList.py`

```
aGrid = []
noRows = int(input("Number rows: "))
noColumns = int(input("Number columns: "))
#Create list
for r in range (0,noRows,1):
    aGrid.append ([])
    for c in range (0,noColumns,1):
        aGrid[r].append("*")
#Display list
for r in range (0,noRows,1):
   for c in range (0,noColumns,1):
      print(aGrid[r][c], end="")
   print()
```

Reading File Information Into A List

- If the amount of information stored in the file can vary then a the list must be dynamically created (using the append() function to add new rows and elements onto the row).
- Input file: chess.txt
  - The starting positions for the program will reside in this file in the form of a simple (unformatted) text file.
  - Each line in the file will represent a row in the chess board.
  - Chess pieces are represented by various characters: PRKBQK
  - Empty locations are represented by a space.

James Tam

Reading File Information Into A List: Display()

- **Name of the example program**: chess.py

```
NEWLINE = "\n"

def display(aBoard,numRows,numColumns):
    currentRow = 0
    currentColumn = 0
    print("DISPLAY BOARD")
    while (currentRow < numRows):
        currentColumn = 0
        while (currentColumn < numColumns):
            print("%s" %(aBoard[currentRow][currentColumn]),end="")
            currentColumn = currentColumn + 1
        currentRow = currentRow + 1
        print()
    for currentColumn in range (0,numColumns,1):
        print("*", end="")
    print(NEWLINE)
```

James Tam

Reading File Information Into A List: Display Grid

```
def displayWithGrid(aBoard,numRows,numColumns):
    currentRow = 0
    currentColumn = 0
    print("DISPLAY BOARD WITH GRID")
    while (currentRow < numRows):
        for currentColumn in range (0,numColumns,1):
            print(" -", end="")
        print()
        currentColumn = 0
        while (currentColumn < numColumns):
            print("|%s" %(aBoard[currentRow][currentColumn]),end="")
            currentColumn = currentColumn + 1
        currentRow = currentRow + 1
        print("|")
    for currentColumn in range (0,numColumns,1):
        print(" -", end="")
```

James Tam

Reading File Information Into A List: File input

```
def readBoardFromFile():
    inputFileOK = False
    aBoard = []
```

James Tam

Reading File Information Into A List: File input (2)

```
# Case: no problems reading from file
while (inputFileOK == False):
    try:
        inputFileName = input("Enter name of input file: ")
        inputFile = open(inputFileName,"r")
        print("Opening file "+ inputFileName + \
              " for reading.")
        currentRow = 0
        for line in inputFile:
             aBoard.append([])
             currentColumn = 0
             for ch in line:
                 if (ch != NEWLINE):
                     aBoard[currentRow].append(ch)
             currentRow = currentRow + 1
        inputFileOK = True
        print("Completed reading of file " + inputFileName)
```

James Tam

Reading File Information Into A List: File input (3)

```
    # Case: file input problems have occurred.
    except IOError:
        print("Error: File", inputFileName, "couldn't" + \
              "be opened.")

# Case: After file input completed.
numRows = len(aBoard)
numColumns = len(aBoard[0])
return(aBoard,numRows,numColumns)
```

James Tam

Reading File Information Into A List: `start()`

```
def start():
    aBoard,numRows,numColumns = readBoardFromFile()
    display(aBoard,numRows,numColumns)
    displayWithGrid(aBoard,numRows,numColumns)

start()
```

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
- How to create a 2D list of variable size and with non-homogenous elements.
- 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
- How to read file information into a dynamically created list