# Files and Exceptions

# File

- Files are suitable for storing results that are needed for an extended period of time, and for holding input values for a programme that will run several times.
- Files are classified as text file or binary file

- Text files: Can only contains characters using an encoding system such as ASCII and UTF-8. can be viewed and modified using any text editor.

- Binary files: sequence of bits. Can contain any type of data, not just characters. E.g. images, sound, videos.

# Opening a File

- A file must be opened before data values can be read from it and before new data value are written to it.

- Files are opened by calling the `open` function.

- The `open` function takes two arguments: The string that contains the name of the file and the string that indicates the *`access mode`* for the file.

- Access modes: read (denoted by "`r`"), write (denoted by "`w`") and append (denoted by "`a`")

```
inf = open("input.txt", "r")
```

- A *`file object`* is returned by the open function.

- Once a file is opened, methods can be applied to the file object to read data from the file or to write data to the file.

- The file should be closed once all of the values have been read or written using the `close` function.

# Reading Input from a File

Our list: numbers

- Can only read from a file when it is in read mode otherwise it crashes if it is in any other mode.

- `Readline` method reads one line from the file and returns it as a string, much like the input function reads a line of text typed on the keyboard.

- `Readline` returns an empty string when there is no further data to read from the file.

er.py | sum_geometric.py | turtle.py | random_values.py

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 20 14:18:00 2019

@author: mireilla
"""

#inf = open("input.txt", "r")
# Read the file name from the user and open the file
fname = input("Enter the file name : ")
inf = open(fname, "r")

# Initialise total
total = 0

# Total the values in the file
line =inf.readline()
while line != "":
    total = total + float(line)
    line = inf.readline()

# Close the file
inf.close()

# Display the result
print("The total of the values in", fname, "is", total)
```

numbers

File Edit

```
12
34
56
78
90
46
788
5
43
64
987
567
94
125
6784567
23
432
12
906
584
368
250
2357
```

```
Enter the file name : numbers.txt
The total of the values in numbers.txt is 6792488.0
```

# Reading Input from a File

- Sometimes it is helpful to read all of the data from the file instead of reading one line at a time..
- The `read` method or the `readline`s method can help accomplish that.
- The `read` method will return the entire content of the file as one string – further processing is required to break the string into smaller pieces
- Readlines returns a list where each element is one line from the file. Once all the lines have been read, a loop can be used to process each element in the list.

```python
# Read the file name from the user and open the file
fname = input("Enter the file name : ")
inf = open(fname, "r")

# Initialise total and read all of the lines from the file
total = 0
lines = inf.readlines()

# Total the values in the file
for line in lines:
    total = total + float(line)


# Close the file
inf.close()

# Display the result
print("The total of the values in", fname, "is", total)
```

# Writing Output to a File

- When a file is opened in `write` mode, a new empty file is created
- So if that file already existed then it will be deleted and all data in it will be lost.
- Open an existing file in `append` mode to write data at the end of that file.
- If the file opened in `append` mode does not exist, then a new file will be created.
- The `write` method can be used to write data opened either to `write` mode or `append` mode.
- It takes one argument, a string. Convert other value types to string using the `str` function.
- Unlike `print method`, the `write` method does not automatically move to the next line. `\n` denotes end of line marker.

```
# Read the file name from the user and open the file
fname = input("Where will the number be stored? ")
outf = open(fname, "w")

# Read the maximum value that will be written
limit = int(input("What is thge maximum value?" ))


# Write the numbers to the file with one number on each line
for num in range (1, limit + 1):
    outf.write(str(num) + "\n")

# Close the file
outf.close()
```

This programme writes the numbers from 1 up to (including) a number entered by the user to a file

# Reading and Writing to a `.csv` File

- .CSV (Comma Separated Values) is associated with importing and exporting spreadsheets and databases.
- Allows greater control over data
- When opening a `.csv` file, you must specify how that file will be used. The options are:

| Code | Description |
|---|---|
| ‘w’ | Creates a new file and writes to that file. If the file already exist, a new file will be created, overwriting the existing file |
| ‘x’ | Creates a new file and writes to that file. If the file already exits, the programme will crash rather than overwriting it. |
| ‘r’ | Opens for reading only and will not allow you to make any changes. |
| ‘a’ | Opens for writing, appending to the end of the file |

# Reading and Writing to a `.csv` File

```python
# This will allow Python to use the .csv library of commands
import csv
# Create a new file called "Stars.csv"
# overwriting any previous file of that name
file = open("Stars.csv", "w")

# Add a new record
newRecord = "Brian, 73, Taurus\n"
file.write(str(newRecord))

# Close and save
file.close()

# Open Stars.csv file to append (add) some data
file = open("Stars.csv", "a")

# Ask the user to enter the name age and star sign
name = input("Enter name: ")
age = input("Enter age: ")
star = input("Enter star sign: ")

# Append all that in the Stars.csv file
newRecord = name + "," + age + "," + star + "\n"
file.write(str(newRecord))
# Close and save
file.close()

# Open the file in read mode
file = open("Stars.csv", "r")
# and display one record at a time
for row in file:
    print(row)

# Open the file in read mode
file = open("Stars.csv", "r")
# Ask the user to enter the data they are searching for
# It will display all rows that contain that data anywhere in that row
search = input("Enter the data you are searching for: ")
reader = csv.reader(file)
for row in file:
   if search in str(row):
       print(row)
```

```python
file = list(csv.reader(open("Stars.csv")))
tmp = []
for row in file:
    tmp.append(row)
```

A .csv file cannot be altered, only added to. If you need to alter the file, you need to write it to a temporary list. This code reads the original file and write it to a list called "tmp"

```python
file = open("NewStars.csv", "w")
x = 0
for row in tmp:
    newRec = tmp[x][0] + "," + tmp[x][1] + "," +tmp[x][2] + "\n"
    file.write(newRec)
    x = x + 1
file.close()
```

Writes from a list to a new csv file called "NewStars.csv"

# More code example - Writing and reading

```python
# Create a file called "Countries.txt".
# If one exists then it will be overwritten with a blank new file.
file = open("Countries.txt", "w")
# Add add four lines of data to the file
file.write("Congo\n")
file.write("Germany\n")
file.write("Spain\n")
file.write("Mexico\n")

# Close the file allowing the chnages to be saved.
file.close()

# Open the Countries.txt file in "read" mode and display the entire file
file = open("Countries.txt", "r")
print(file.read())

# Open the Countries.txt file in "append" mode
file = open("Countries.txt", "a")

# Add another line and then close the file
file.write("Senegal\n")

# Close file
file.close()
file =open("Countries.txt", "r")
# Display file content
print(file.read())
```

```
Congo
Germany
Spain
Mexico

Congo
Germany
Spain
Mexico
Senegal
```

If the file.close() is not included, the changes will not be saved to the text file.

# Command Line Arguments

- Being able to provide input from the command line is beneficial when writing scripts that use multiple programmes to automate some tasks and programmes that are scheduled to run periodically.
- Any command line argument provided when the programme was executed are stored into a variable named `argv` (argument vector) that resides in `sys` (system)

```python
# The system module must be imported to access the command line arguments
import sys

# Display the number of command line arguments (including the .py file)
print("The programme has ", len(sys.argv), \
      "command line argument(s).")

# Display the name of the .py file
print ("The name of the .py file is", sys.argv[0])

# Determine whether or not there are additional arguments to display
if len(sys.argv) > 1:
    # Display all of the commena line arguments beyond the name of the .py file
    print("The remaining arguments are: ")
    for i in range (1, len(sys.argv)):
        print(" ", sys.argv[1])
else:
    print("There are no additional arguments")
```

This programme demonstrates accessing the argument vector.

```
In [105]: runfile('C:/Users/mireilla/.spyder-py3/command_line_codes.py', wdir='C:
The programme has  1 command line argument(s).
The name of the .py file is C:/Users/mireilla/.spyder-py3/command_line_codes.py
There are no additional arguments
```

# Command Line Arguments

- Command line arguments can be used to supply any input values to the programme that can be typed on the command line such as integers, floating-point numbers, and strings

```python
# Import the system module
import sys

#Ensure that the programme was started with on command line argument beyond
# the name of the .py file
if len(sys.argv) != 2:
    print("A file name must be porvided as a command line",\
          "argument.")
    quit()

# Open the file listyed immediately after the .py file on the command line
inf = open(sys.argv[1], "r")

# Initialiase the total
total = 0

#Total the values in the file
line = inf.readline()
while line != "":
    total = total + float(line)
    line = inf.readline()


# Close the file
inf.close()

#Display the result
print("The total of the values in", sys.argv[1], "is", total)
```

# Exceptions

- `Exceptions` are those errors that the user can make. For example, a user can supply a non-numeric value when a numeric value was expected.
- By default a Python programme will crash when an `exception` occurs.
- So the programmer must indicate where an `exception` might occur in order to catch it.
- The programmer must also indicate what code to run to handle the `exception` when it happens.
- `Try` and `except` are used.
- The code that might cause an exception that we want to catch is place inside the `try` block
- The `try` block is immediately followed by one or more `except` blocks.
- When an `exception` occurs inside a `try` block, execution immediately jump to the appropriate `except` block without running any remaining statements in the `try` block.
- An `except` block that does not specify a particular `exception` will catch any type of `exception` (that is not caught by another `except` block associated to the same `try` block.
- The `except` block only execute when an `exception` occurs.

# Exceptions example code

- Quits when the file requested by the user is not found which is not always ideal.

```
# Read the file name from the user
fname = input("Enter the file name: ")

# Attempt to open the file
try:
    inf = open(fname, "r")
except FileNotFoundError:
    # Display an error message and quit if the file was not opened successfully
    print("'%s' could not be opened, Quitting...")
    quit()
```

A user can re-enter the name of the file. However this can also cause an exception. A loop is used to run until the user enters the name of the file successfully. The try and except are inside the loop

```
# Read the file name from the user
fname = input("Enter the file name: ")

file_opened = False
while file_opened == False:
    # Attempt to open file
    try:
        inf = open(fname, "r")
        file_opened = True
    except FileNotFoundError:
        # Display an error message and quit if the file was not opened successf
        print("'%s' could not be opened, Quitting...")
        fname = input("Enter file name: ")
```