# CS50 W6 - File I/O

**File I/O** is the ability of a program to take a file as input or create a file as output.

names = [ ]

for \_ in range(3):

name = input("What's your name? ")

names.append(name)

names = [ ]

for \_ in range(3):

names.append(input("What's your name? "))

names = [ ]

for \_ in range(3):

names.append(input("What's your name? "))

for name in sorted(names):

print(f"hello, {name} ")

\*The sorted() function sorts the list in alphabetical order. Notice that once this program is executed, all information is lost. File I/O allows our program to store this information such that it can be used later.

## 

## `open`

open is a functionality built into Python that allows you to open a file and utilize it in your program.

1. **“w”**

name = input("What's your name? ")

file = open(“names.txt”, “w”)

file.write(name)

file.close()

\*The open() function opens a file called names.txt with writing enabled “w”. The opened file is assigned to a variable called file.The line file.write(name) writes the name to the text file and the file is close().

\*Note that if you run the program multiple times using different names, it will completely rewrite the names.txt from scratch.

1. **“a”**

name = input("What's your name? ")

file = open(“names.txt”, “a”)

file.write(name)

file.close()

\*Changing “w” (writing) to “a” (appending) will allow us to continuously add names to the file. But there’s a problem, the names will be appended without any gaps between them (LazerAkivaEden).

1. **\n**

name = input("What's your name? ")

file = open(“names.txt”, “a”)

file.write(**f**“**{**name**}\n**”)

file.close()

\*Adding a new line **“\n”** after each of the names will solve this issue.

## `with`

It is quite easy to forget to close the file. Using a with block allows us to automate the closing of a file.

1. **“a”**

name = input("What's your name? ")

with open(“names.txt”, “a”) as file:

file.write(**f**“**{**name**}\n**”)

\*Notice that the line below the with block is indented.

1. **“r”**

with open(“names.txt”, “r”) as file:

lines = file.readlines()

for line in lines :

print(“hello ”, line)

\*Notice that readlines() has the ability to read all the lines of a file and store them in a file called lines. Also, notice that readlines() automatically adds a line break. This, added to print() line break ends up with double line breaks

```

hello, Lazer

hello, Akiva

```

1. rstrip()

with open(“names.txt”, “r”) as file:

lines = file.readlines()

for line in lines :

print(“hello ”, line.rstrip())

\*Adding rstrip() to the print() function has the effect of removing the extra line break at the end of each line.

```

hello, Lazer

hello, Akiva

```

1. Simplified

with open(“names.txt”, “r”) as file:

for line in file :

print(“hello ”, line.rstrip())

\*This version uses `for line in file: ` which removes the need to readlines() and iterate through all of them before printing the result.

1. sorted()

names = [ ]

with open(“names.txt”) as file:

for line in file :

names.append(line.rstrip())

for name in sorted(names) :

print(f“hello, {name}”)

\*Notice that we initialize an empty list called names in memory, which will hold the names we append(). Then each name is sorted() before it is printed. To sort the list in reverse order we pass in a second argument to sorted(names, reverse=True)

1. Sorting the file

with open(“names.txt”) as file:

for line in sorted(file) :

print(f“hello, ”, line.rstrip())

\*In this simplified version we sorted() the file directly. Notice also we did not have to specify “w” since it is the default mode to open a file.

## CSV

**csv** “comma separated values”

1. split()

with open("students.csv") as file:

for line in file:

row = line.rstrip().split(",")

print(f"**{**row[0]**}** is in **{**row[1]**}**")

\*rstrip() removes the end of each line in our CSV file. split() tells the compiler where to find the end of our values in our CSV file. row[0] is the first element of each line, row[1] the second.

with open("students.csv") as file:

for line in file:

name, house = line.rstrip().split(",")

print(f"**{**name**}** is in **{**house**}**")

\*Instead of assigning our operation to the variable row, we can assign two variables `name, house`.

1. sorted()

students = [ ]

with open("students.csv") as file:

for line in file:

name, house = line.rstrip().split(",")

students.append(f"**{**name**}** is in **{**house**}**")

for student in sorted(students):

print(student)

\*Here we create a list called students and append() each string to this list. Then we output a sorted() version of the list.

1. Dictionary

students = [ ]

with open("students.csv") as file:

for line in file:

name, house = line.rstrip().split(",")

student = {}

student[“name”] = name

student[“house”] = house

students.append(student)

for student in students:

print(f"**{**student[‘name’]**}** is in **{**student[‘house’]**}**")

\*In this version we initialize a dictionary `student{}` and we add the values for each student as a key-value pair. Then we append() each dictionary to the students list. Notice that in the f-string we are using a combination of single and double quotes to differentiate.

students = [ ]

with open("students.csv") as file:

for line in file:

name, house = line.rstrip().split(",")

student = {“name”: name, “house”: house}

students.append(student)

for student in students:

print(f"**{**student[‘name’]**}** is in **{**student[‘house’]**}**")

\*Notice the simplified `student = {“name”: name, “house”: house}`.

1. key=

students = [ ]

with open("students.csv") as file:

for line in file:

name, house = line.rstrip().split(",")

student = {“name”: name, “house”: house}

students.append(student)

def get\_name(student):

return student[‘name’]

for student in sorted(students, key=get\_name):

print(f"**{**student[‘name’]**}** is in **{**student[‘house’]**}**")

\*Notice that sorted() needs to know how to get the `key` of each student. We define the get\_name() function which simply returns `student[“name”]` and add it as a parameter `key=get\_name` to the sorted() function.

\*We could also reverse the order `for student in sorted(students, key=get\_name, reverse=True):`

1. lambda()

students = [ ]

with open("students.csv") as file:

for line in file:

name, house = line.rstrip().split(",")

students.append({“name”: name, “house”: house})

for student in sorted(students, key=lambda student: student[“name”]):

print(f"**{**student[‘name’]**}** is in **{**student[‘house’]**}**")

\*A lambda function is an anonymous, inline function defined using the lambda keyword. In this code it is used as a key to the sorted() function. It takes the `student` dictionary as an argument and returns the value associated with the `name`key. This value is then used to sort the dictionaries in the list.

1. csv module

import csv

students = [ ]

with open("students.csv") as file:

reader = csv.reader(file)

for name, home in reader:

students.append(**{**“name”: name, “home”: home**}**)

for student in sorted(students, key=lambda student: student[“name”]):

print(f"**{**student[‘name’]**}** is in **{**student[‘house’]**}**")

\*reader() function from the csv module automatically manages the reading and separating of values.

1. DictReader, reader

import csv

students = [ ]

with open("students.csv") as file:

reader = csv.DictReader(file)

for row in reader:

students.append(**{**“name”: name, “home”: home**}**)

for student in sorted(students, key=lambda student: student[“name”]):

print(f"**{**student[‘name’]**}** is in **{**student[‘house’]**}**")

\*First we need to update the csv file to contain headers:

```

name,home

Harry,“Number Four Privet Drive”

Ron,The Burrow

Draco, Malfoy Manor

```

\*In this code we replaced reader by DictReader, which returns one dictionary at a time. This makes the code more robust as long as the CSV file has the correct header information.

1. writer

import csv

name = input("what’s your name? ")

home = input("where’s your home? ")

with open("students.csv", “a”) as file:

writer = csv.writer(file)

writer.writerow([name, home])

\*The csv.writer() function takes the file as an argument. writerow() takes the line that we want to write to the file as a list [name, home].

\*At first, the original csv file only contain the headers name,home inputting a name and home the csv file will now show:

```

name,home

Harry,“Number Four Privet Drive”

```

1. DictReader, writer

import csv

name = input("what’s your name? ")

home = input("where’s your home? ")

with open("students.csv", “a”) as file:

writer = csv.DictWriter(file, fieldnames=**[**“name”, “home”**]**)

writer.writerow(**{**“name”: name, “home”: home**}**)

\*The csv.DictWriter() function takes two parameters: the file, and `fieldnames=[“name”, “home”]`, which is telling the compiler to write a row with two fields called name and home. writerow() now takes a dictionary as a parameter `**{**“name”: name, “home”: home**}**`.

## Binary Files and `PIL`

A binary file is simply a collection of ones and zeros. This type of file can store anything including audio and image data.

PIL “pillow” is a Python imaging library that adds image processing capabilities.

1. **costumes.py**

import sys

from PIL import Image

images = [ ]

for arg in sys.argv[1:]:

image = Image.open(arg)

images.append(image)

images[0].save(

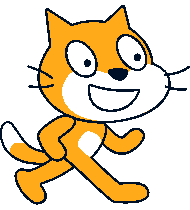
"costumes.gif", save\_all=True, append\_images=[images[1]], duration=200, loop=0

)

\*In this code we import the sys module to be able to use command-line arguments. We also import the Image functionality from PIL. The for loop simply loops through the images provided as command-line arguments and stores them into the list called image and [1:] starts slicing argv at its second element.

\*In the last line of code: “costumes.gif” is the name of the file to be created. save\_all=True saves all of the frames passed to the library. append\_images=[image[1]] appends the second image to the list already containing image[0]. duration=200 specifies the duration of each frame to 200 milliseconds. loop=0 determines that the loop will be infinite.

\*This code results in an animated GIF (costumes.gif) that consists of two frames, each displayed for 200 milliseconds, looping forever:



1. **image.py**

from PIL import Image

def main():

with open("in.jpeg") as img:

print(img.size)

print(img.format)

main()

1. **rotate()**

from PIL import Image

def main():

with open("in.jpeg") as img:

img = img.rotate(180)

img.save(”out.jpeg”)

main()

1. **filter()**

from PIL import Image

def main():

with open("in.jpeg") as img:

img.filter(ImageFilter.BLUR)

img.save(”out.jpeg”)

main()

1. **filter()**

from PIL import Image

def main():

with open("in.jpeg") as img:

img.filter(ImageFilter.FIND\_EDGES)

img.save(”out.jpeg”)

main()

1. **views.py**, **DictReader**, **PIL**

import csv

def main():

with open("views.csv", “r") as file:

reader = csv.DictReader(file)

for row in reader:

print(row)

main()

\*This code will print the entire row in dict format using headers. Specifying the key, like in row[“id”] would print all data under `id` column.

1. **NumPy**, **PIL**

import csv

import numpy as np

from PIL import Image

def main():

with open("views.csv", “r") as file:

reader = csv.DictReader(file)

for row in reader:

brightness = calculate\_brightness(f"**{**row[‘id’]**}**.jpeg")

print(round(brightness))

def calculate\_brightness(filename):

with Image.open(filename) as image:

brightness = np.mean(np.array(img.convert(”L”))) / 255

return brightness

main()

\*main() calls the calculate\_brightness() function of every image associated with each “id” key in the csv file and prints out a resulting rounded value for each image.

\*calculate\_birghtness() takes row[‘id’] as an argument, Image.open the image file associated with the ‘id’ and calculates its brightness:

* img.convert(“L”) method is used to convert the image to grayscale, where “L” stands for luminance.
* np.array() is used to convert the grayscale image into a NumPy array. Each element of this array represents the brightness of a pixel (0 black to 255 white).
* np.mean() calculates the average value of the elements in the NumPy array, which corresponds to the average brightness of the image.
* Finally, the average brightness is divided by 255 to normalize the value to a range between 0 and 1.



import csv

import numpy as np

from PIL import Image

def main():

# Open two csv files, one in reading mode and one for writing the results

with open("views.csv", “r") as views, open("analysis.csv", “w") as analysis:

reader = csv.DictReader(views)

# Create a new object containing the operation writing operation on `analysis`

# Specify analysis.csv should have the same fieldnames (headers) than views.csv

# And one more header “brightness”

writer = csv.DictWriter(

analysis, fieldnames=reader.fieldnames + ["brightness"]

)

# Write the headers to analysis.csv

writer.writeheader()

for row in reader:

brightness = calculate\_brightness(f"**{**row[‘id’]**}**.jpeg")

# Write the rest of the rows (as dict) to analysis.csv

# Assign data from reader to analysis.csv keys + brightness

writer.writerow(

**{**

“id”: row[“id”]

“english\_title”: row[“english\_title”]

“japanese\_title”: row[“japanese\_title”]

“brightness”: round(brightness, 2)

**}**

)

def calculate\_brightness(filename):

with Image.open(filename) as image:

brightness = np.mean(np.array(img.convert(”L”))) / 255

return brightness

if \_\_name\_\_ == “\_\_main\_\_”

main()

1. Optimize for loop

# Instead of writing a new dictionary, we can simply add “brightness” key

# To `row` dictionary before writing it and shortcut the process

for row in reader:

row[“brightness”] = round(calculate\_brightness(f"**{**row[‘id’]**}**.jpeg"), 2)

writer.writerow(row)