### Module 8.2 JSON Practice

Spring 2025 CSD325 Advanced Python

**Author**: Brittaney Perry-Morgan

**Date**: Sunday, June 29th, 2025

Module 8.2 JSON Practice

**main.py**

"""

Name: Brittaney Perry-Morgan

Date: Sunday, June 29th, 2025

Assignment: Module 8.2 JSON Practice

Purpose: Implementation of a student management system.

Imports:

- json: Used to interact with the JSON file.

- sys: Used to add the project root to the Python path.

- Path: Used to work with file paths.

"""

*import* json

*import* os

*import* sys

*import* tkinter as tk

from pathlib *import* Path

from tkinter *import* messagebox

*# Add project root to the Python path to*

project\_root = Path(\_\_file\_\_).resolve().parents[1]

sys.path.append(str(project\_root))

*# flake8: noqa: E402*

from module\_8.student *import* ( *# pylint: disable=wrong-import-position*

Student,

StudentList,

)

JSON\_FILE\_PATH = Path(\_\_file\_\_).parent / "data" / "student.json"

def load\_students(*file\_path*: *Path*) -> StudentList:

"""

Loads students from JSON (PascalCase *keys*). Removes existing duplicates.

Parameters:

- file\_path: The path to the JSON file.

:type file\_path: *Path*

Returns:

- A StudentList containing the loaded students.

:rtype: *StudentList*

"""

*students\_data* = []

if *file\_path*.exists() and *file\_path*.stat().st\_size > 0:

with open(*file\_path*, *encoding*="utf-8", *mode*="r") as file:

try:

*students\_data* = json.load(file)

except json.JSONDecodeError:

print(

f"Warning: JSON file at {*file\_path*} is empty or malformed. \n"

f"Starting with an empty list."

)

*students\_data* = []

*students* = [

Student(

*f\_name*=item["F\_Name"],

*l\_name*=item["L\_Name"],

*student\_id*=item["Student\_ID"],

*email*=item["Email"],

)

*for* item in students\_data

*if* all(k in item *for* k in ("F\_Name", "L\_Name", "Student\_ID", "Email"))

]

*student\_list* = StudentList(students) *# pylint: disable=redefined-outer-name*

student\_list.remove\_duplicates() # Remove any existing duplicates in the file!

return *student\_list*

# pylint: *disable*=redefined-outer-name

def save\_students(*file\_path*: *Path*, *student\_list*: *StudentList*) -> None:

"""

Save students to JSON with PascalCase keys.

Parameters:

- file\_path: The path to the JSON file.

:type file\_path: *Path*

- student\_list: The StudentList to save.

:type student\_list: *StudentList*

"""

*file\_path*.parent.mkdir(*parents*=True, *exist\_ok*=True)

*data\_to\_save* = [

{

"F\_Name": s.f\_name,

"L\_Name": s.l\_name,

"Student\_ID": s.student\_id,

"Email": s.email,

}

*for* s in *student\_list*.students

]

try:

with open(*file\_path*, *encoding*="utf-8", *mode*="w") as file:

json.dump(*data\_to\_save*, *file*, *indent*=4)

except (*IOError*, *OSError*) as e:

print(f"Error saving students to {*file\_path*}: {e}")

*raise*

def student\_notification(*user\_msg*: *str*) -> str:

"""

Returns the user message if it's not *empty*, otherwise returns a default message.

Parameters:

- user\_msg: The user message to return.

:type user\_msg: *str*

Returns:

- The user message if it's not *empty*, otherwise returns a default message.

:rtype: *str*

"""

return *user\_msg* or "Invalid input. Please try again..."

def get\_relative\_path(*path*: *Path*) -> str:

"""

Returns a relative path string from the current working directory to the given path.

Parameters:

- path: The path to get the relative path for.

:type path: *Path*

Returns:

- The relative path string.

:rtype: *str*

"""

try:

return os.path.relpath(str(*path*), *start*=os.getcwd())

except ValueError as e:

print(f"Error getting relative path: {e}")

return str(*path*)

def show\_save\_dialog(*file\_path*: *Path*) -> bool:

"""

Shows a dialog asking the user if they want to save the changes.

Parameters:

- file\_path: The path to the file being modified.

:type file\_path: *Path*

Returns:

- True if the user clicks 'Yes', False otherwise.

:rtype: *bool*

"""

*root* = tk.Tk()

root.withdraw() # Hide the main *window*

*relative\_path* = get\_relative\_path(*file\_path*)

*message* = f"This file has been modified outside. Do you want to reload it?\n\n{relative\_path}"

return messagebox.askyesno("File Modified", *message*)

if \_\_name\_\_ == "\_\_main\_\_":

JSON\_FILE\_PATH.parent.mkdir(*parents*=True, *exist\_ok*=True)

*student\_list* = load\_students(JSON\_FILE\_PATH)

print(student\_notification(f"\n{'=' \* 50}\nOriginal Student List:\n{'=' \* 50}\n"))

student\_list.print\_students()

print("\n")

*new\_student* = Student(

*f\_name*="Brittaney",

*l\_name*="Perry-Morgan",

*student\_id*=12345,

*email*="bperrymorgan@me.com",

)

if student\_list.contains\_student(*new\_student*):

print(

student\_notification(

f"\n\*\*\*\*\* DUPLICATE STUDENT DETECTED: ({new\_student}) will not be added.\*\*\*\*\*\n\n"

)

)

*else*:

student\_list.add\_student(new\_student)

print(

student\_notification(

f"\n\*\*\*\*\* STUDENT ADDED: ({new\_student}) has been added.\*\*\*\*\*\n\n"

)

)

*if* show\_save\_dialog(JSON\_FILE\_PATH):

save\_students(JSON\_FILE\_PATH, student\_list)

print(student\_notification("Changes saved."))

*else*:

print(student\_notification("Changes not saved."))

print(student\_notification(f"\n{'=' \* 50}\nUpdated Student List:\n{'=' \* 50}\n"))

student\_list.print\_students()

print("\n")

student\_list = load\_students(JSON\_FILE\_PATH)

print(

student\_notification(

f"\n{'=' \* 50}\nUpdated Student List from JSON:\n{'=' \* 50}\n"

)

)

student\_list.print\_students()

print("\n")

**data/student.json (original)**

[

{

"F\_Name": "Ellen",

"L\_Name": "Ripley",

"Student\_ID": 45604,

"Email": "eripley@gmail.com"

},

{

"F\_Name": "Arthur",

"L\_Name": "Dallas",

"Student\_ID": 45605,

"Email": "adallas@gmail.com"

},

{

"F\_Name": "Joan",

"L\_Name": "Lambert",

"Student\_ID": 45714,

"Email": "jlambert@gmail.com"

},

{

"F\_Name": "Thomas",

"L\_Name": "Kane",

"Student\_ID": 68554,

"Email": "tkane@gmail.com"

}

]

**data/student.json (updated)**

[

{

"F\_Name": "Ellen",

"L\_Name": "Ripley",

"Student\_ID": 45604,

"Email": "eripley@gmail.com"

},

{

"F\_Name": "Arthur",

"L\_Name": "Dallas",

"Student\_ID": 45605,

"Email": "adallas@gmail.com"

},

{

"F\_Name": "Joan",

"L\_Name": "Lambert",

"Student\_ID": 45714,

"Email": "jlambert@gmail.com"

},

{

"F\_Name": "Thomas",

"L\_Name": "Kane",

"Student\_ID": 68554,

"Email": "tkane@gmail.com"

},

{

"F\_Name": "Brittaney",

"L\_Name": "Perry-Morgan",

"Student\_ID": 12345,

"Email": "bperrymorgan@me.com"

}

]

**student.py**

"""

Name: Brittaney Perry-Morgan

Date: Sunday, June 1st, 2025

Assignment: Module 8.2 JSON

Purpose: Holds the Student and StudentList dataclasses.

Imports:

- dataclass: Used to create dataclasses.

- field: Used to create default factory for the students list.

- List: Used to type hint the students list.

"""

from dataclasses *import* dataclass, field

from typing *import* List

@dataclass

class Student:

"""

Representation of a Student.

Fields:

- f\_name: The first name of the student.

:type f\_name: str

- l\_name: The last name of the student.

:type l\_name: str

- student\_id: The student's unique ID.

:type student\_id: int

- email: The student's email address.

:type email: str

"""

f\_name: str

l\_name: str

student\_id: int

email: str

def \_\_str\_\_(*self*) -> str:

"""String representation of a student."""

return f"{*self*.l\_name}, {*self*.f\_name} : *ID* = {*self*.student\_id}, *Email* = {*self*.email}"

@*dataclass*

class StudentList:

"""

Representation of a list of students.

Fields:

- students: The list of students.

:type students: List[Student]

"""

students: List[Student] = field(*default\_factory*=list)

def \_\_iter\_\_(*self*):

"""Iterator for the StudentList."""

*return* iter(*self*.students)

def print\_students(*self*) -> None:

"""Print all students in the list."""

for student in *self*.students:

print(*student*)

def add\_student(*self*, *student*: *Student*) -> None:

"""

Add a student to the list.

Parameters:

- student: The Student to add to the list.

:type student: *Student*

"""

*self*.students.append(*student*)

def contains\_student(*self*, *student*: "Student") -> bool:

"""

Check for duplicate by student\_id OR email.

Parameters:

- student: The student to check for duplicates.

:type student: *Student*

Returns:

- True if the student is a *duplicate*, False otherwise.

:rtype: *bool*

"""

return any(

s.student\_id == *student*.student\_id or s.email == *student*.*email*

for s in *self*.*students*

)

def remove\_duplicates(*self*) -> None:

"""

Removes duplicate students by student\_id or *email*, keeping the first occurrence.

"""

*seen\_ids* = set()

*seen\_emails* = set()

*unique\_students* = []

for s in *self*.students:

if s.student\_id not in seen\_ids and s.email not in seen\_emails:

unique\_students.append(*s*)

seen\_ids.add(s.*student\_id*)

seen\_emails.add(s.*email*)

*self*.*students* = unique\_students

**deliverables**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**