tabulate, Virtual Environments, PyPI & the AiCE Student Report

AiCE Software Engineering Foundations — Python Programming (Advanced)

This closing lesson on external libraries introduces tabulate for professional table output, virtual environments for isolating a project's dependencies, and a tour of PyPI's most useful packages — then closes with a capstone combining colorama and tabulate into one polished report.

1. tabulate — Displaying Data in Formatted Tables

tabulate is a library for formatting Python data as text tables. It takes a list of data and a list of headers and produces a clean, aligned table — the kind you'd expect from a professional command-line application. Without it, printing tabular data requires careful manual spacing with f-strings, and still often looks inconsistent.

```
from tabulate import tabulate

students = [
    ["Amara Diallo", "Senegal", 88, "A", "PASS"],
    ["Kofi Mensah", "Ghana", 45, "F", "FAIL"],
    ["Ngozi Adeyemi", "Nigeria", 92, "A", "PASS"],
]

headers = ["Name", "Country", "Average", "Grade", "Status"]

# Grid style
print(tabulate(students, headers=headers, tablefmt="grid"))

# Pretty style
print(tabulate(students, headers=headers, tablefmt="pretty"))
```

Format	Description	Best For
plain	No borders, aligned spacing	Minimal, clean output
simple	Dashes under headers	Most common default style
grid	Full grid with + at intersections	Maximum clarity and readability
pretty	Rounded corners with padding	Polished, professional look
pipe	Markdown table format	Copy into Markdown documentation

tabulate With a List of Dictionaries

When your data is a list of dictionaries — common when loading from JSON or building student records — tabulate can use the dictionary keys as headers automatically:

```
from tabulate import tabulate

students = [
    {"Name": "Amara Diallo", "Country": "Senegal", "Score":
88, "Grade": "A"},
    {"Name": "Kofi Mensah", "Country": "Ghana", "Score": 45,
"Grade": "F"},
]

# headers="keys" tells tabulate to use dictionary keys as
column headers
print(tabulate(students, headers="keys", tablefmt="grid"))
```

tabulate automatically right-aligns numbers, left-aligns text, and sizes each column to its widest value — this is the exact table format used in the Student Management System capstone.

Combining colorama and tabulate

Used together, colorama supplies the color and tabulate supplies the structure — exactly the combination the capstone project relies on:

```
import colorama
from colorama import Fore, Style
from tabulate import tabulate
colorama.init()

def colour(text, col, bold=False):
    s = Style.BRIGHT if bold else ""
    return f"{s}{col}{text}{Style.RESET_ALL}"

students = [
    {"Name": "Amara Diallo", "Grade": "A", "Status": "PASS"},
    {"Name": "Kofi Mensah",  "Grade": "F", "Status": "FAIL"},
]

coloured_data = []
for s in students:
    row = {
        "Name"  : s["Name"],
        "Grade" : colour(s["Grade"], Fore.GREEN if s["Grade"]
== "A" else Fore.RED, bold=True),
        "Status": colour(s["Status"], Fore.GREEN if
s["Status"] == "PASS" else Fore.RED, bold=True),
    }
    coloured_data.append(row)

print(tabulate(coloured_data, headers="keys",
tablefmt="grid"))
```

2. Virtual Environments — An Important Concept

When you install packages with pip3, they're installed globally — available to every Python program on your system. For simple projects and learning, this is fine. For professional projects, there's a real risk: two projects may need different versions of the same library.

Virtual environments solve this by creating isolated Python environments — each project gets its own set of installed packages, independent of everything else:

```
# Create a virtual environment in a folder called 'venv'
python3 -m venv venv

# Activate it (Linux/macOS)
source venv/bin/activate

# Your prompt will change to show (venv) at the beginning
# Now install packages — they go into the venv, not globally
pip install colorama tabulate

# Deactivate when done
deactivate
```

For this course and the Student Management System capstone, the global installation already done is perfectly adequate. But as you move into real projects afterward, using a virtual environment for every project is the professional standard.

> **Inside a Virtual Environment, Use pip, Not pip3**
> *Note that inside an activated virtual environment, you use pip (not pip3), because the environment already knows which Python to use.*
>
> **requirements.txt in Professional Projects**
> *When working with virtual environments, always generate a requirements.txt after installing your dependencies: pip freeze > requirements.txt. This captures the exact versions of every installed package. Anyone who clones your repository can then run pip install -r requirements.txt to recreate your exact environment. This file belongs in your GitHub repository alongside your code.*

3. Exploring PyPI — Finding Libraries

pypi.org is the official home of Python packages. Any time you have a task that feels like something that must have been solved before — sending emails, generating PDFs, making HTTP requests, parsing HTML, working with spreadsheets — search PyPI first.

Domain	Package	What It Does
HTTP requests	requests	Make HTTP GET/POST requests to APIs
Data analysis	pandas	DataFrames for structured data manipulation
Machine learning	scikit-learn	Classification, regression, clustering
Web framework	flask / django	Lightweight or full-featured web applications
Terminal tables	tabulate	Format data as clean text tables
Terminal colour	colorama	Cross-platform terminal colour output
Image processing	pillow	Open, edit, and save image files
PDF generation	reportlab	Create PDF documents programmatically
Database ORM	sqlalchemy	Database access with Python objects
Excel files	openpyxl	Read and write Excel .xlsx files
Testing	pytest	Professional testing framework for Python

Each of these is one pip3 install command away — this is the ecosystem you now have access to, built over decades by developers across the globe, including a growing number of African developers contributing to open source every day.

4. Bringing It All Together — The AiCE Student Report

This unit's capstone brings colorama and tabulate together in a polished student report program:

```
import colorama
from colorama import Fore, Style
from tabulate import tabulate
colorama.init()

def c(text, col, bold=False):
```

```
    s = Style.BRIGHT if bold else ""
    return f"{s}{col}{text}{Style.RESET_ALL}"

green = lambda t, b=False: c(t, Fore.GREEN, b)
red = lambda t, b=False: c(t, Fore.RED, b)
yellow = lambda t, b=False: c(t, Fore.YELLOW, b)
cyan = lambda t, b=False: c(t, Fore.CYAN, b)

raw_students = [
    {"name": "Amara Diallo", "country": "Senegal", "math": 88,
"english": 76, "science": 91},
    {"name": "Kofi Mensah",  "country": "Ghana",   "math": 45,
"english": 52, "science": 38},
    {"name": "Ngozi Adeyemi","country": "Nigeria", "math": 92,
"english": 85, "science": 88},
]

def get_grade(avg):
    return "A" if avg >= 70 else "B" if avg >= 60 else "C" if
avg >= 50 else "F"

table_data = []
all_averages = []
for s in raw_students:
    avg = round((s["math"] + s["english"] + s["science"]) / 3,
1)
    grade = get_grade(avg)
    status = "PASS" if avg >= 50 else "FAIL"
    all_averages.append(avg)
    table_data.append({
        "Name": s["name"], "Country": s["country"], "Average":
avg,
        "Grade": green(grade, True) if grade in ("A","B") else
red(grade, True),
        "Status": green(status, True) if status == "PASS" else
red(status, True)
    })

print(cyan("=" * 60, True))
print(cyan(" AiCE STUDENT PERFORMANCE REPORT", True))
```

```
print(cyan("=" * 60, True))
print(tabulate(table_data, headers="keys", tablefmt="grid"))

class_avg = round(sum(all_averages) / len(all_averages), 2)
passed = sum(1 for a in all_averages if a >= 50)
print(f"\nClass average: {yellow(str(class_avg), True)}")
print(f"Passed: {green(str(passed), True)} /
{len(raw_students)}")
```

The output is a fully formatted, color-coded student report — exactly the kind of output used in the Student Management System capstone ahead. The combination of colorama for color and tabulate for tables produces terminal output that is genuinely professional.

> **Build What Does Not Already Exist**
>
> *"The ability to build on other people's work — to use a library that took months to build in one import statement — is one of the great privileges of the open source era. Do not rebuild what already exists. Build what does not."*
>
> *— Adesina Olagunju, Founder, AiCE*

Key Takeaways

tabulate turns raw lists or dictionaries into clean, aligned tables with a single function call, and combining it with colorama's color helpers produces genuinely professional terminal output. Virtual environments isolate a project's dependencies from every other project on the same machine — the professional standard once you move beyond this course — and a requirements.txt file, generated with pip freeze, is how a project's exact dependencies travel with its code. With external libraries covered, the next unit introduces JSON, the universal data format used by web APIs everywhere, and the natural next step after storing data in CSV files.