Choosing the Right Data Structure & the Course Enrollment Analyser

AiCE Software Engineering Foundations — Python Programming (Intermediate)

This closing lesson brings together every data structure covered across the beginner and intermediate sections into one clear comparison, then closes both this unit and the entire Python Intermediate section with a capstone that puts tuples and sets to work side by side.

1. Tuples vs. Lists vs. Sets vs. Dictionaries — Complete Comparison

Feature	List	Tuple	Set	Dictionary
Ordered?	Yes	Yes	No	Yes (insertion order)
Mutable?	Yes	No	Yes	Yes
Allows duplicates?	Yes	Yes	No	Keys: No, Values: Yes
Access by index?	Yes	Yes	No	No (by key)
Access by key?	No	No	No	Yes
Syntax	[a, b]	(a, b)	{a, b}	{k: v}
Use when	Ordered, changeable collection	Fixed record or return value	Unique items or set math	Named fields / key lookups

The Four Questions

Does order matter? A list or tuple. Should the data ever change? If yes, list or dictionary; if no, tuple. Do you need uniqueness or set math? A set. Do you need to look things up by name? A dictionary. Answer these four questions honestly about your data, and the right structure becomes obvious.

2. Bringing It All Together — The AiCE Course Enrollment Analyser

This unit's capstone — and the final program of the Python Intermediate section — uses both tuples and sets together in a realistic student enrollment analysis system:

```
# AiCE Course Enrollment Analyser
# Uses tuples for fixed records and sets for enrollment
analysis
from collections import namedtuple

# Define an immutable student record using namedtuple
Student = namedtuple("Student", ["id", "name", "country",
"courses"])

# Student data — courses field is a frozenset (immutable set)
students = [
    Student("AiCE001", "Amara Diallo",     "Senegal",
frozenset(["Python", "SQL", "Git"])),
    Student("AiCE002", "Kofi Mensah",       "Ghana",
frozenset(["Python", "JavaScript"])),
    Student("AiCE003", "Ngozi Adeyemi",     "Nigeria",
frozenset(["SQL", "Git", "Linux"])),
    Student("AiCE004", "Fatima Al-Hassan",  "Sudan",
frozenset(["Python", "SQL"])),
    Student("AiCE005", "Kwame Asante",      "Ghana",
frozenset(["JavaScript", "Git", "Linux"])),
    Student("AiCE006", "Zainab Musa",       "Nigeria",
frozenset(["Python", "JavaScript", "SQL"])),
    Student("AiCE007", "Diallo Bah",        "Guinea",
frozenset(["Python", "Git"])),
]

# All available courses
all_courses = {"Python", "SQL", "JavaScript", "Git", "Linux"}

# Build enrollment sets per course using a set comprehension
course_enrollment = {
```

```
    course: {s.name for s in students if course in s.courses}
    for course in all_courses
}

# Build country sets
countries = {s.country for s in students}
country_counts = {
    country: sum(1 for s in students if s.country == country)
    for country in countries
}

# Students taking Python AND SQL
py_and_sql = course_enrollment["Python"] &
course_enrollment["SQL"]

# Students taking Python but NOT SQL
py_not_sql = course_enrollment["Python"] -
course_enrollment["SQL"]

# Display the report
print("=" * 55)
print(" AiCE COURSE ENROLLMENT ANALYSIS REPORT")
print("=" * 55)

print("\nCourse Enrollment:")
for course in sorted(all_courses):
    enrolled = sorted(course_enrollment[course])
    print(f"  {course:<15}: {len(enrolled)} students —
{enrolled}")

print("\nCountry Breakdown:")
for country, count in sorted(country_counts.items(),
key=lambda x: -x[1]):
    print(f"  {country:<15}: {count} student(s)")

print(f"\nStudents in Python AND SQL: {sorted(py_and_sql)}")
print(f"Students in Python NOT SQL: {sorted(py_not_sql)}")

print("\nIndividual Student Profiles:")
for s in students:
```

```
    sid, name, country, courses = s    # Tuple unpacking
    missing = all_courses - courses    # Courses not taken
    print(f"  [{sid}] {name:<22} | {country:<15} | Enrolled:
{len(courses)} | Missing: {sorted(missing)}")
print("=" * 55)
```

This program demonstrates the full power of tuples and sets working together:

namedtuple creates immutable, readable student records.

frozenset creates an immutable set of courses per student, suitable as a namedtuple field.

Set comprehensions build enrollment lists per course.

Set operations (intersection, difference) answer enrollment questions instantly.

Tuple unpacking reads each student record cleanly.

This is the level of thinking that characterizes intermediate-to-advanced Python — choosing the right data structure for each piece of information, and composing them to produce clear, efficient code.

3. Closing the Python Intermediate Section

Across this section, you have transformed from a beginner who writes sequential scripts into a developer who writes organized, modular, resilient, and expressive Python programs. The concepts you now command:

Functions — named, reusable blocks with parameters, defaults, and return values.

Scope and return values — understanding where data lives and how it flows between functions.

Modules and import — leveraging Python's standard library and building your own modules.

File handling — reading, writing, and appending data that persists between sessions.

Exception handling — catching errors gracefully and building resilient programs.

List comprehensions — concise, expressive list creation and transformation.

Tuples and sets — immutability and uniqueness as tools for clearer, safer code.

> **What Comes Next**
> *"The intermediate programmer is not someone who knows more commands. They are someone who thinks differently about problems — who sees functions where others see repetition, who sees structure where others see chaos, who handles failure before it arrives. That is you now. Build on it." — Adesina Olagunju, Founder, AiCE*

Key Takeaways

Four questions settle almost any choice between Python's core data structures: does order matter, should the data change, do you need uniqueness or set math, and do you need lookup by name. The Course Enrollment Analyser shows tuples and sets working side by side with everything else from this section — functions, comprehensions, and structured data — to produce a genuinely useful analysis tool from a small amount of code. This closes the Python Intermediate section: functions, scope, modules, file handling, exception handling, list comprehensions, and tuples and sets now form a complete, professional foundation to build on.