**Day 2 Labs – Python I: Data Types, Object Model & I/O**

**Lab 1: Python Truthiness, Object Identity & Type System**

**Objective:**

* Understand Python's object model, mutability, and truthy/falsy behavior

**Steps:**

1. Open Python shell or Jupyter Notebook.
2. Test truthy/falsy values:

python

values = [None, 0, 0.0, '', [], {}, set(), 'False', 1, [0], True]

for val in values:

print(f"{repr(val)} is {bool(val)}")

1. Inspect object identity:

python

a = [1, 2]

b = a

c = list(a)

print(id(a), id(b), id(c))

print(a is b) # True

print(a == c) # True

print(a is c) # False

1. Practice with immutables:

python

x = 10

def modify(val):

val += 5

return val

print(modify(x)) # 15

print(x) # Still 10

**Outcome:**

* Clear understanding of identity (is), equality (==), and mutability

**Lab 2: Reading Large CSVs – Full Load vs Chunked Processing**

**Objective:**

* Compare memory and performance of full CSV load vs chunked processing

**Steps:**

1. Download a large sample CSV (~50MB+):

bash

wget https://people.sc.fsu.edu/~jburkardt/data/csv/hw\_200.csv -O data.csv

1. Load full CSV using pandas:

python

import pandas as pd

import time

start = time.time()

df = pd.read\_csv('data.csv')

print(df.shape)

print("Full load time:", time.time() - start)

1. Load with chunked reader:

python

start = time.time()

reader = pd.read\_csv('data.csv', chunksize=10000)

row\_count = 0

for chunk in reader:

row\_count += len(chunk)

print("Row count:", row\_count)

print("Chunked read time:", time.time() - start)

1. Monitor memory usage (optional with memory\_profiler):

bash

pip install memory-profiler

**Outcome:**

* Learn trade-offs between full memory load and chunk-based streaming

**Lab 3: CSV to Parquet with Structured Logging & CLI**

**Objective:**

* Build a small ETL CLI tool using argparse that reads CSV and writes Parquet with logging

**Steps:**

1. Create etl.py:

python

import argparse

import pandas as pd

import logging

import os

logging.basicConfig(level=logging.INFO, format='%(asctime)s [%(levelname)s] %(message)s')

def convert\_csv\_to\_parquet(input\_file, output\_file):

logging.info(f"Reading CSV: {input\_file}")

df = pd.read\_csv(input\_file)

logging.info(f"Writing to Parquet: {output\_file}")

df.to\_parquet(output\_file, index=False)

logging.info("Conversion complete")

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(description='Convert CSV to Parquet')

parser.add\_argument('--infile', required=True, help='Input CSV path')

parser.add\_argument('--outfile', required=True, help='Output Parquet path')

args = parser.parse\_args()

if not os.path.exists(args.infile):

logging.error("Input file not found")

else:

convert\_csv\_to\_parquet(args.infile, args.outfile)

1. Run the tool:

bash

python etl.py --infile data.csv --outfile output.parquet

1. Inspect output Parquet (optional):

python

import pandas as pd

df = pd.read\_parquet("output.parquet")

print(df.head())

**Outcome:**

* Practical CLI tool using standard Python modules for argument parsing, logging, and file conversion

**Completion Checklist:**

* Practiced Python identity and mutability behavior
* Compared full vs chunked read performance
* Built a CLI CSV → Parquet converter with structured logging