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)}")
```

3. 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
```

4. 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
```

2. 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)
```

3. 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)
```

4. 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)
```

2. Run the tool:

bash

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

3. 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