Python programming Lab(23CP301P)

Name: Krishika Vansh

Semester: V

Roll No: 23BCP448

Faculty: Mr. Davinder Singh

Division: VII Batch G13

Branch: Computer Engineering

School of Technology

November 2025

Experiment No: 7

Implementing Logging Mechanism in Python

Objective: To analyze a Python-based project, identify potential points for logging, and implement dummy code with logging at key places such as function calls, exception handling, input/output operations, and loops for debugging and monitoring purposes.

Code:

```
import logging
import json
import random
import time
import os

logging.basicConfig(
   level=logging.DEBUG,
   format='%(asctime)s | %(levelname)-7s | %(message)s'
)
logger = logging.getLogger("simple")

def read_input(path):
   logger.info("Reading input file: %s", path)
   if not os.path.exists(path):
      logger.info("Input not found — writing a small sample to %s", path)
```

```
    sample = {"name": "sample-run", "values": [random.random() for _ in range(200)], "factor": 1.1}
    with open(path, "w") as f:
      json.dump(sample, f)
  try:
    with open(path) as f:
      data = json.load(f)
    logger.debug("Loaded JSON keys: %s", list(data.keys()))
    return data
  except Exception:
    logger.exception("Failed to read/parse input")
    return None

def validate_input(data):
  logger.info("Validating input")
  if not data:
    logger.warning("No data provided")
    return False
  if "values" not in data or not isinstance(data["values"], list):
    logger.error("Missing or invalid 'values' field")
    return False
  logger.debug("Validation OK (name=%s, n_values=%d)", data.get("name"), len(data["values"]))
  return True

def process_values(values):
```

```
    logger.info("Processing %d values", len(values))
    result = []
    for i, v in enumerate(values):

        if i % 50 == 0 and i > 0:
            logger.debug("Processed %d items", i)
        try:
            num = float(v)

            if num < 0.2:
                logger.debug("Small value at index %d: %f", i, num)
            result.append(num * 2)
        except Exception:
            logger.warning("Skipping invalid item at index %d: %r", i, v)
    logger.info("Processing finished, kept %d items", len(result))
    return result

def fake_api_call(payload):
    logger.info("Calling external service with %d items",
len(payload.get("items", [])))
    try:
        time.sleep(random.uniform(0.01, 0.05))
        if random.random() < 0.05:
            raise TimeoutError("simulated timeout")
        resp = {"status": "ok", "received": len(payload.get("items", []))}
```

```
        logger.info("External service replied: %s", resp)
        return resp
    except Exception:
        logger.exception("External call failed")
        return {"status": "error"}

def main(input_path="input.json"):
    logger.info("Run started")
    data = read_input(input_path)
    if not validate_input(data):
        logger.error("Validation failed, exiting")
        return

    values = process_values(data["values"])
    if not values:
        logger.error("No valid values after processing, exiting")
        return

    payload = {"items": values[:100], "meta": {"name": data.get("name")}}
    resp = fake_api_call(payload)

    if resp.get("status") == "ok":
        logger.info("Pipeline succeeded for %s", data.get("name"))
    else:
```

```
    logger.warning("Pipeline finished with issues (status=%s)",
resp.get("status"))


  logger.info("Run finished")


if __name__ == "__main__":
  main()
```

Output: