Python Worksheets

Below is a list of core concepts along with practical tasks that engineers can perform to validate and assess their Python skills for everyday work-related scenarios:

---

1. Basic Syntax and Data Types

- **Concepts:** Variables, basic data types (integers, floats, strings), and control structures (if-else, loops).
- **Task:**
  - Write a script that reads a text file and counts the occurrence of each word.
  - **Validation:** Verify that the output dictionary correctly represents word counts for given test files.

---

2. File I/O Operations

- **Concepts:** Reading from and writing to files, handling file paths, and using context managers.
- **Task:**
  - Create a script that reads a log file, filters specific lines (e.g., containing "ERROR"), and writes the filtered content to a new file.
  - **Validation:** Check that the output file contains only the lines that meet the filter criteria.

---

3. String Manipulation

- **Concepts:** String methods, slicing, formatting, and parsing.
- **Task:**
  - Develop a script that processes log entries by extracting the timestamp, severity, and message from each line.
  - **Validation:** Ensure that the parsed elements match the expected values for sample log entries.

---

4. Regular Expressions (Optional)

- **Concepts:** Pattern matching using Python's `re` module for search, match, and substitution tasks.
- **Task:**
  - Write a function to extract email addresses or specific patterns from a block of text using regular expressions.

    - **Validation:** Provide several test cases to confirm that the function correctly identifies or rejects entries.

---

5. Exception Handling (Optional)

- **Concepts:** Try-except blocks, raising exceptions, and creating custom error messages.
- **Task:**
    - Create a script that attempts to open a file and gracefully handles the case when the file is missing, logging an appropriate error message.
    - **Validation:** Simulate a missing file scenario and confirm that the error is caught and logged as expected.

---

6. Functions and Modular Programming

- **Concepts:** Function definitions, parameters, return values, and basic modular design.
- **Task:**
    - Refactor one of the earlier tasks (such as log parsing) into reusable functions and modules.
    - **Validation:** Test the functions independently to ensure they work correctly with predefined inputs.

---

7. Data Structures: Lists, Dictionaries, and Sets

- **Concepts:** Efficient use of lists, dictionaries, tuples, and sets to store and manipulate data.
- **Task:**
    - Given a list of log lines, group them by severity (e.g., INFO, WARNING, ERROR) using a dictionary.
    - **Validation:** Verify that the dictionary keys and counts correctly reflect the distribution of log entries.

---

8. List and Dictionary Comprehensions

- **Concepts:** Using comprehensions for concise and efficient data transformations.
- **Task:**
    - Write a script that filters out and transforms a list of log entries using list comprehensions (for instance, extracting only error messages).
    - **Validation:** Confirm that the resultant list contains only the expected entries.

---

9. Command-line Argument Parsing (Optional)

- **Concepts:** Utilizing modules like `argparse` for handling command-line arguments.
- **Task:**

    - Develop a script that accepts a file name and a search term as command-line arguments, processes the file, and prints matching lines.
    - **Validation:** Run the script with various arguments to ensure that the output changes as expected.

---

10. Basic Unit Testing (Optional)

- **Concepts:** Using Python’s `unittest` or `pytest` modules to write and run test cases.
- **Task:**
    - Write unit tests for one or more of the functions developed (e.g., log parsing or regex matching).
    - **Validation:** Ensure that all tests pass when the functions are correct and that failures are appropriately flagged when introducing errors.

These tasks are designed to simulate everyday challenges that engineers face, such as analyzing logs and executing simple automation scripts. By completing these worksheets, engineers can self-evaluate their strengths and identify areas for further improvement in Python.

Task list

Task 1: Word Count Script
Covers: Basic syntax, data types, and file I/O
Description: Read a text file, count the frequency of each word, and output the results as a dictionary.

Task 2: Log Filter
Covers: File I/O and exception handling
Description: Process a log file to extract lines containing "ERROR" and write them to a new file, handling missing files gracefully.

Task 3: Log Parser
Covers: String manipulation and regular expressions
Description: Extract and format key details (timestamp, severity, message) from each log entry.

Task 4: Modular Code Development
Covers: Functions, modular programming, and data structures
Description: Refactor the log parser into reusable functions, and group log entries by severity using dictionaries.

Task 5: Command-Line Tool
Covers: Command-line argument parsing and list comprehensions
Description: Create a script that accepts file names and search terms from the command line, processes logs, and outputs filtered results.

Task 6: Unit Testing
Covers: Basic unit testing
Description: Write tests using a framework like `unittest` or `pytest` to validate the functionality of the previously developed scripts.

Task 7: CSV Data Analyzer

- **Concepts:** File I/O, data structures (lists and dictionaries), list comprehensions, and basic data analysis.
- **Description:**
  Develop a script that reads a CSV file containing columns such as "TestCase," "Status," and "ExecutionTime." The script should produce a summary report showing counts for each status (e.g., Passed, Failed) and compute the average execution time.
- **Validation:**
  Use a sample CSV file and compare the output summary with expected statistics.

Task 8: JSON Configuration Updater

- **Concepts:** File I/O, JSON processing, dictionary manipulation, and exception handling.
- **Description:**
  Create a script that loads a JSON configuration file, updates specific keys based on given conditions (for example, updating a version number or toggling a feature flag), and writes the modified configuration back to disk.

- **Validation:**
  Test the script with a sample JSON file and ensure only the intended keys are modified while the rest of the configuration remains intact.

Task 9: Directory Log Aggregator

- **Concepts:** File I/O, directory traversal (using modules such as `os` or `glob`), string manipulation, and aggregation using dictionaries.
- **Description:**
  Write a script that scans a specified directory for all `.log` files, processes each file, and aggregates counts of different log levels (e.g., INFO, WARNING, ERROR) into a single summary dictionary.
- **Validation:**
  Place several sample log files in a test directory and verify that the aggregated counts correctly reflect the combined data from all files.

Task 10: Data Format Converter (CSV to JSON)

- **Concepts:** File I/O, CSV and JSON modules, data transformation, and list comprehensions.
- **Description:**
  Develop a script that reads data from a CSV file and converts each row into a JSON object. The final output should be a JSON file containing an array of these objects.
- **Validation:**
  Provide a sample CSV file and confirm that the output JSON file accurately represents the CSV content structure.

Task 11: Nested Data Structure Flattener

- **Concepts:** Handling nested dictionaries/lists, recursion or iterative approaches, and data transformation using comprehensions.
- **Description:**
  Create a script that accepts a nested dictionary (for example, loaded from a JSON file) and flattens it into a single-level dictionary. Use a separator (like an underscore) to concatenate nested keys into new single keys.
- **Validation:**
  Use a sample nested structure to verify that the output dictionary correctly maps all nested values to single-level keys.

Task 12: Custom Log Filter with Criteria

- **Concepts:** Advanced string manipulation, regular expressions, file I/O, and dictionary usage.

- **Description:**
  Write a script that reads a log file and filters entries based on custom criteria—such as logs within a specific time range or containing a particular keyword pattern. Employ regex to accurately extract and process these elements.
- **Validation:**
  Run the script on sample log data and confirm that the filtered output matches the specified criteria.