

5.2 Ejercicio de programación 2

Francisco Natanael Ortiz Martínez

A01796014

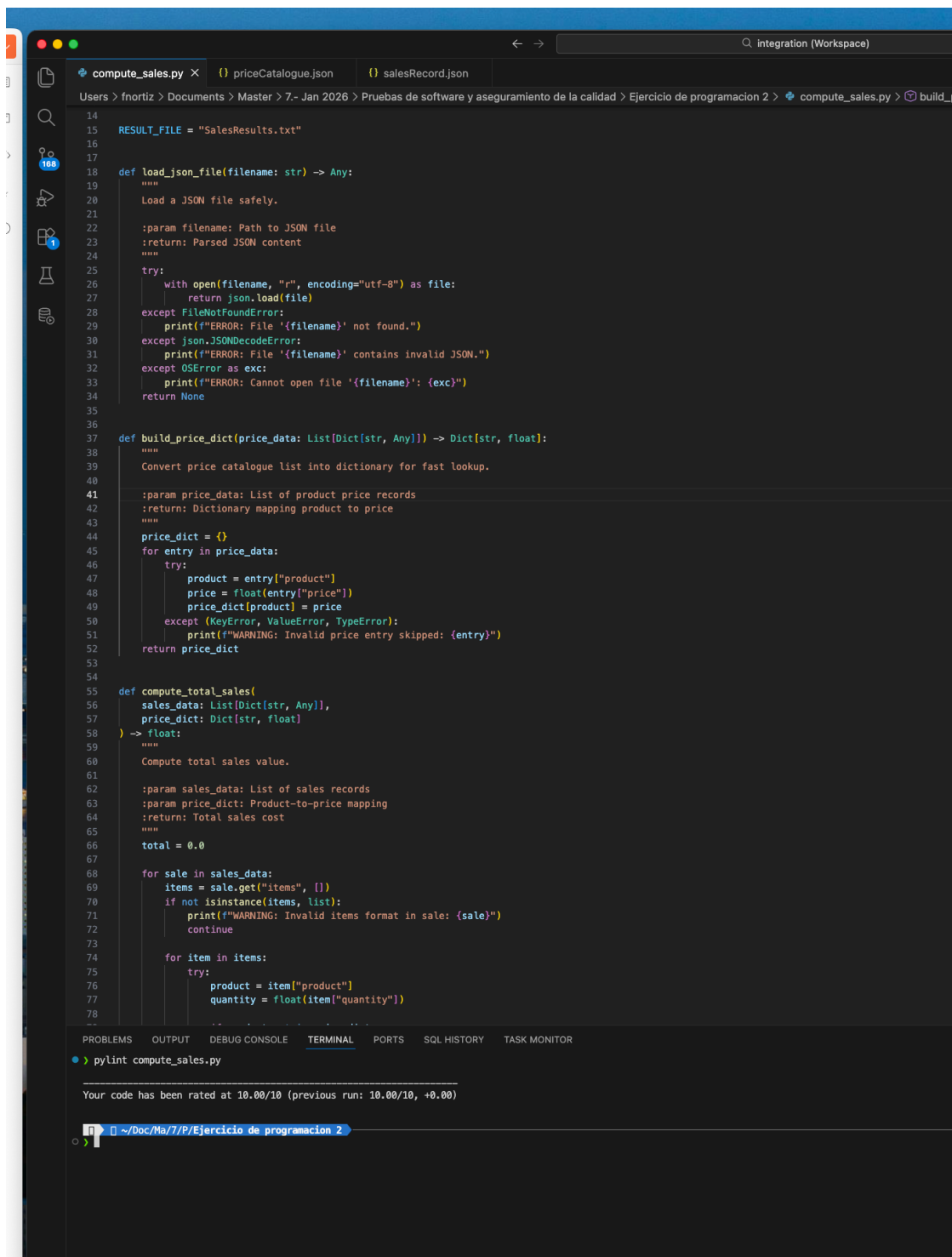
Introduction

This activity focuses on the design and implementation of a command-line Python application named `computeSales.py`, which processes structured business data in JSON format. The program reads a product price catalogue and a company sales record, computes the total cost of all sales transactions, and produces a clear, human-readable report both on the console and in an output file. Beyond functional correctness, the solution emphasizes robustness, scalability, and reliability, ensuring that invalid or inconsistent data is handled gracefully without interrupting execution.

A central objective of this activity is adherence to professional coding standards. The implementation strictly follows PEP 8, the Python Enhancement Proposal that defines conventions for formatting, naming, code organization, spacing, and readability. Writing code that complies with PEP 8 ensures consistency, maintainability, and clarity qualities that are essential in real-world software development environments. Proper use of docstrings, meaningful variable names, structured functions, and controlled line length contribute to a clean and readable codebase that can be easily understood and extended by other developers.

In addition to style compliance, the program is validated using static analysis tools such as `flake8` and `pylint`. These tools enforce coding discipline by detecting stylistic issues, potential bugs, unused variables, overly complex logic, and deviations from best practices. By ensuring the code passes `flake8` checks and achieves a high `pylint` score, the activity demonstrates a commitment not only to functionality but also to code quality and maintainability. This standards-driven approach reflects professional software engineering practices and prepares the solution for scalability and long-term evolution.

Evidence of executions



The image shows a code editor window with a dark theme. The editor displays a Python file named `compute_sales.py`. The code includes a function `load_json_file` for safely loading JSON files, a function `build_price_dict` for converting a list of price records into a dictionary, and a function `compute_total_sales` for calculating the total sales value from a list of sales records. The code uses type hints and includes error handling for file operations and data parsing.

```
14
15 RESULT_FILE = "SalesResults.txt"
16
17
18 def load_json_file(filename: str) -> Any:
19     """
20     Load a JSON file safely.
21
22     :param filename: Path to JSON file
23     :return: Parsed JSON content
24     """
25     try:
26         with open(filename, "r", encoding="utf-8") as file:
27             return json.load(file)
28     except FileNotFoundError:
29         print(f"ERROR: File '{filename}' not found.")
30     except json.JSONDecodeError:
31         print(f"ERROR: File '{filename}' contains invalid JSON.")
32     except OSError as exc:
33         print(f"ERROR: Cannot open file '{filename}': {exc}")
34     return None
35
36
37 def build_price_dict(price_data: List[Dict[str, Any]]) -> Dict[str, float]:
38     """
39     Convert price catalogue list into dictionary for fast lookup.
40
41     :param price_data: List of product price records
42     :return: Dictionary mapping product to price
43     """
44     price_dict = {}
45     for entry in price_data:
46         try:
47             product = entry["product"]
48             price = float(entry["price"])
49             price_dict[product] = price
50         except (KeyError, ValueError, TypeError):
51             print(f"WARNING: Invalid price entry skipped: {entry}")
52     return price_dict
53
54
55 def compute_total_sales(
56     sales_data: List[Dict[str, Any]],
57     price_dict: Dict[str, float]
58 ) -> float:
59     """
60     Compute total sales value.
61
62     :param sales_data: List of sales records
63     :param price_dict: Product-to-price mapping
64     :return: Total sales cost
65     """
66     total = 0.0
67
68     for sale in sales_data:
69         items = sale.get("items", [])
70         if not isinstance(items, list):
71             print(f"WARNING: Invalid items format in sale: {sale}")
72             continue
73
74         for item in items:
75             try:
76                 product = item["product"]
77                 quantity = float(item["quantity"])
78
```

Below the code editor, there is a terminal window. The terminal shows the command `pylint compute_sales.py` being executed. The output indicates that the code has been rated at 10.00/10, which is the same as the previous run.

```
pylint compute_sales.py

Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

