4-1 Journal: Explore Data Structures

Marie-Chantal Foster

Cybersecurity, Southern New Hampshire University

CS-500-10018-M01 Introduction to Programming

Dr. Shadha Tabatabai

December 7, 2025

Lists, Tuples, and Dictionaries

Lists, tuples, and dictionaries are core data structures in Python and the building blocks for organizing and managing information. Each offers unique properties in a program, and we will address them. I chose a practical application in a shopping cart scenario that involves managing products, quantities, and prices, which require thoughtful data organization. We will discuss the application, advantages, disadvantages, and challenges faced with these data structures. Below is a brief overview of the data structures(*Differences and applications* 2025).

Features	Lists	Tuples	Dictionaries
Mutability	Mutable container (the elements can change and grow or shrink).	Immutable(elements can not change).	Mutable (entries can be added, modified or removed)
Order	Elements maintain their insertion order	Ordered	Ordered (Python 3.7 version)
Indexing	Integer-indexed. Sequence of different types: strings, integers, floats or other lists	Integer-indexed. Sequence types: len(), indexing and other sequence functions.	Key- value pairs
Example use	Storing items that change and defined using [ ] square brackets	Fixed configurations or coordinates and defined using ( ) parenthesis	Mapping keys to values and defined using { } curly braces

Lists are most commonly used in Python, and in this scenario, they are essential because they provide ordered, changeable collections of data (ZyBooks, 2022). A *list* in use case can manage the current inventory of products as the user shops, for instance: `cart = [“Apples”, “Pineapples”, “Oranges”]`. *Advantages*: *Lists* allow items to be mutable, making it easy to add or remove items as the user shops, enhancing flexibility, for example, in a `cart.remove(“Bananas”)`(GeeksforGeeks, 2021). Plus, lists preserve the order in which items are added, giving a structured list. *Disadvantages*: They can experience performance bottlenecks, for example, when a user is searching for “Oranges”, a linear search would occur to check each item one by one, and when there are thousands of items, it can be inefficient. Since

items can grow significantly, the cart could become overloaded and consume more memory, resulting in substantial memory overhead (Ramos, 2025). *Challenges:* Assigning one list to another (`new_cart = old_cart`) creates a reference, not a new list; modifying `new_cart` also changes `old_cart.` To avoid that, a copy must be created using the `slice ([:])` or the `copy()` method (*Reference vs Copy,* 2025).

Tuples are not as widely used as *lists*, but they are ideally suited when the element's relative position is important and must remain unchanged, thereby maintaining the data's integrity (ZyBooks, 2022). The applicability of product order details can be represented as a single fixed product record in a database. For instance, ("SKU123", "White Bread", 2.99). Once the item is defined in the product catalog, the information should not be changed by the shopping cart logic. *Advantages:* Data integrity is crucial to preventing errors in financial calculations and inventory tracking; therefore, the item's price or SKU is write-protected. Also, *tuples* generally perform slightly faster when processing and iterating over *lists* because their immutable structure is optimized more effectively (Ramos, 2025). *Disadvantages*: When updates are needed, the entire *tuple* object must be replaced, not just the element within it. For example, if changing "White Bread" to "Wheat Bread", the first word would not be replaced, but instead the entire element. Also, there are no built-in in-place modification functions, such as `.sort()`, which adds extra steps (Ramos, 2025). *Challenges*: Syntax errors cause issues when storing product data. A single value in parentheses, `sku=(123),` is treated as an integer, but a trailing comma is required to define the single-element *tuple* `sku=(123,)`. Another challenge is the lack of semantic clarity because their elements can only be accessed by integer index (e.g., `fruit[0], fruit[1]`). Without meaningful keys, it is not clear what each position represents, which can make the code difficult to understand and maintain long-term (Pena, 2025)

The third data structure is *Dictionaries.* They are great at storing data as key-value pairs, which provide fast lookups and well-organized data association, ideal for the shopping cart scenario. Using an actual shopping cart allows items to be tracked by a unique identifier (key) and associated with details (value) (ZyBooks, 2022). *Advantages:* Instantaneous and retrieval access to look up, insert, and delete an item's details; e.g., the user can search for `cart["grocery_sku_123"]` without searching the entire list. They are logical and have semantic organization, e.g., key-value pairs, so instead of remembering the arbitrary numerical indices such as `grocery[0]`, the user can use meaningful strings as keys like `grocery["Wheat Bread"]`(Jesse, 2024). *Disadvantages:* In version 3.7, ordering would have been unreliable, leading to grocery items being displayed inconsistently in checkout summaries. They also require more memory because they need to store hash values and reserve space for collisions, especially in a large cart, e.g., *lists* or *tuples* rely on numeric indices (`item[2]`), while *dictionaries* allow (`cart_dict["Apples"]['price']`) (Jesse, 2024). *Challenges:* They can encounter issues with missing keys; if the cart processing code attempts to look up a key that is no longer present, it will result in a `KeyError,` crashing the checkout process unless properly handled. Another challenge is the lack of enforced structure, which can lead to inconsistent data. For instance, if one part of the cart stores `price` as a float, while another stores `cost` as a string, and then the script is run while trying to sum all the prices, then the script could fail because the keys are not consistent or the data types do not match (Ramos, 2024).

In conclusion, I learned the differences data structures make in a program, and when applied in a shopping cart application, I can see which ones have strengths and weaknesses. Yet I also discovered that tuples are easy to understand because of their simplicity and immutability, whereas lists and dictionaries are ideal for their flexibility and dynamism.

References

Differences and applications of list, tuple, set, and dictionary in Python. (2025, October 3). GeeksforGeeks. https://www.geeksforgeeks.org/python/differences-and-applications-of-list-tuple-set-and-dictionary-in-python/

Jesse. (2024, July 2). Dictionaries vs Lists in Python and Why it is important? Medium. https://medium.com/pythons-gurus/dictionaries-vs-lists-in-python-and-why-it-is-important-a7a5d74a99a3

Pena, A. (2025, February 25). *Python lists vs. tuples: Understanding their differences and best uses*. Udacity Blog. https://www.udacity.com/blog/2025/02/python-lists-vs-tuples-understanding-their-differences-and-best-uses.html

Python Variable Assignment: Reference vs Copy. (2025, January 20). coderivers. https://coderivers.org/blog/python-variable-assignment-reference-vs-copy/

Ramos, L. (2024, December 16). *Dictionaries in Python*. Real Python. https://realpython.com/python-dicts/

Ramos, L. (2025, January 26). *Lists vs Tuples*. Real Python. https://realpython.com/python-lists-tuples/

zyBooks. (2022). Introduction to programming. Wiley. http://www.zybooks.com