Data Structures

Python includes several standard programming data structures, such as `list`, `tuple`, `dict`, and `set`, as part of its built-in types. Many applications do not require other structures, but when they do, the standard library provides powerful and well-tested versions that are ready to be used.

The `enum` module provides an implementation of an *enumeration* type, with iteration and comparison capabilities. It can be used to create well-defined symbols for values, instead of using literal strings or integers.

The `collections` module includes implementations of several data structures that extend those found in other modules. For example, `Deque` is a double-ended queue, which allows the addition or removal of items from either end. The `defaultdict` is a dictionary that responds with a default value if a key is missing, while `OrderedDict` remembers the sequence in which items are added to it. `namedtuple` extends the normal `tuple` to give each member item an attribute name in addition to a numeric index.

For large amounts of data, an `array` may make more efficient use of memory than a `list`. Since the `array` is limited to a single data type, it can use a more compact memory representation than a general-purpose `list`. At the same time, `array` instances can be manipulated using many of the same methods as a `list`, so it may be possible to replace a `list` with an `array` in an application without a lot of other changes.

Sorting items in a sequence is a fundamental aspect of data manipulation. Python's `list` includes a `sort()` method, but sometimes it is more efficient to maintain a list in sorted order without re-sorting it each time its contents are changed. The functions in `heapq` modify the contents of a list while preserving the sort order of the list with low overhead.

Another option for building sorted lists or arrays is `bisect`. It uses a binary search to find the insertion point for new items, and is an alternative to repeatedly sorting a list that changes frequently.

Although the built-in `list` can simulate a queue using the `insert()` and `pop()` methods, it is not thread-safe. For true ordered communication between threads use the `queue` module. `multiprocessing` includes a version of a `Queue` that works between processes, making it easier to convert a multithreaded program to use processes instead.

`struct` is useful for decoding data from another application, perhaps coming from a binary file or stream of data, into Python's native types for easier manipulation.

This chapter covers two modules related to memory management. For highly interconnected data structures, such as graphs and trees, use `weakref` to maintain references while still allowing the garbage collector to clean up objects after they are no longer needed. Use the functions in `copy` for duplicating data structures and their contents, including making recursive copies with `deepcopy()`.

Debugging data structures can be time consuming, especially when wading through printed output of large sequences or dictionaries. Use `pprint` to create easy-to-read representations that can be printed to the console or written to a log file for easier debugging.

Finally, if the available types do not meet the requirements, subclass one of the native types and customize it, or build a new container type using one of the abstract base classes defined in `collections` as a starting point.

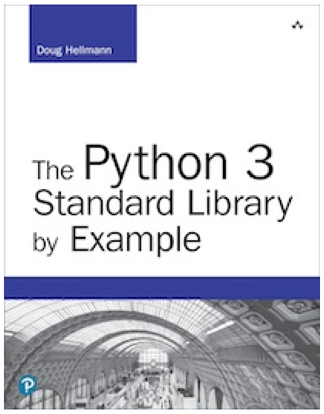
- enum – Enumeration Type
- collections — Container Data Types
- array — Sequence of Fixed-type Data
- heapq – Heap Sort Algorithm
- bisect — Maintain Lists in Sorted Order
- queue — Thread-Safe FIFO Implementation
- struct — Binary Data Structures
- weakref — Impermanent References to Objects
- copy — Duplicate Objects
- pprint — Pretty-Print Data Structures

difflib — Compare Sequences   enum – Enumeration Type

This page was last updated 2017-01-28.

Navigation

difflib — Compare Sequences
enum – Enumeration Type



The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?