

CHAPTER 6: STRUCTURES

A *structure* is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. (Structures are called "records" in some languages, most notably Pascal.)

While we talk about "data structures", and how to use them in every language, this section is about understanding how software developers carefully control the low-level "shape" of their data items to solve their problems.

When you first learn about the C `struct` keyword, you might think that it is equivalent to a Python `dict` - a dynamic key-value store like a PHP array, Java map, or JavaScript object - but nothing is further from the truth. These other languages provide us with easy-to use data structures where all of the challenging problems are solved.

This chapter tells the creators of Python, PHP, Java, and JavaScript *how* to solve the complex problems and build convenient and flexible data structures which we use in all of those object-oriented languages.

One way to look at the code in this chapter is to think of it as a lesson on how one might build Python's `list` and `dict` data structures. If the code in the chapter takes you a little while to figure out - mentally make a note of thanks for all the hard work that the modern languages invest to make their higher-level data structures so flexible and easy to use.

The traditional example of a structure is the payroll record: an "employee" is described by a set of attributes such as name, address, social security number, salary, etc. Some of these in turn could be structures: a name has several components, as does an address and even a salary.

Structures help to organize complicated data, particularly in large programs, because in many situations they permit a group of related variables to be treated as a unit instead of as separate entities. In this chapter we will try to illustrate how structures are used. The programs we will use are bigger than many of the others in the book, but still of modest size.

6.1 Basics

Let us revisit the date conversion routines of [Chapter 5](#). A date consists of several parts, such as the day, month, and year, and perhaps the day of the year and the month name. These five variables can all be placed into a single structure like this:

```
struct date {
    int day;
    int month;
    int year;
    int yearday;
    char mon_name[4];
};
```