Box 2.2 Further Reading

- Rossum, Warsaw, and Coghlan (2024), the general Python style guide
- Rossum, Lehtosalo, and Langa (2024), the original type hint style guide
- Gonzalez et al. (2024), the variable annotation style guide
- Google (2024), the Google Python style guide

2.7 The Design of Programs

A program should be designed. Engineers are designers, so we should make excellent programmers. Unfortunately, many engineers fail to carefully consider the design of their programs, at least when it comes to those programs used for analysis and design. This often leads to programs that function poorly and are difficult to maintain. The costs associated with this are usually much greater than those accrued by a systematic design process.[5]

How should a program be designed? Software design methods abound; however, they are similar to the (many) design methods used for more conventional engineering products. Our familiarity with these, as engineers, may allow a cursory introduction to suffice.

A design typically begins with a **design problem**: a product (i.e., solution) is desired, one that does something (i.e., produces an output) for someone (i.e., a customer). For engineering computing programs, we engineers are often the customers, and the output is usually information for an analysis or design problem. The product or solution is the program itself. The design problem is often rather ill-defined at first, and we must question the customer about their goals throughout the design process. Often, the problem we thought we were solving at the beginning changes throughout the design process. Similarly, the program (solution) undergoes several iterations.

Two of the most important ways to think about program design are introduced in the rest of this section: (1) the functional analysis design method and (2) the pseudocode algorithm representation.

5. It should be noted, however, that for back-of-the-envolope calculations, we need not spend the time on a systematic design process. A paradigm I like to use is to create an exploratory `play.py` file in a project or simply use an interactive session. In this type of environment, we can explore without concern for structure, style, and careful design processes.

2.7.1 The Functional Analysis Design Method

One way to organize our thinking about the program (solution) is to begin at the highest level and ask the question:

> What will the program start with and what will it need to produce?

This amounts to the question: What are its **inputs** and **outputs**? Figure 2.1a illustrates this high-level conception of the program as a block that transforms inputs into outputs.

(a) High-level product functionality. (b) General function.

Figure 2.1. The functional design method (a) at the highest level and (b) in general, for any level.

This is the beginning of the **functional analysis design method**. We have treated the program as a **function**, which, like a mathematical function, maps inputs to ouputs. The next step is to consider the question:

> How can the program achieve this transformation of its inputs to its outputs?

Many techniques may be explored, but often they can be separated into **subfunctions**. The subfunctions can themselves have subfunctions. This way of breaking down the problem into functions and subfunctions is the key to the power of the functional analysis design method. We see that, at all levels, the paradigm of mapping of inputs to outputs through functions applies, as illustrated in figure 2.1b. Drawing these functional blocks and connecting their inputs and outputs is a crucial step in a program design process.

In Python program design, we have Python functions (section 1.9) and methods (section 1.5) to perform the role of the function blocks in the functional analysis design method. Inputs are passed as input arguments (or objects) and outputs are the returned values (or mutated objects). If we begin by sketching a functional diagram of inputs, outputs, and functional blocks from the highest level to the lowest, the programming of the corresponding Python functions and methods becomes a matter of implementing a structure we have already thoroughly considered. This technique is a great way to overcome the anxiety of the "blank page."

2.7.2 Algorithm Representation via Pseudocode

At a certain depth, the functional blocks of section 2.7.1 have reached a level that is best treated as indivisible. It is not always obvious when this point is reached, but we can always iterate later. From here, simple functional blocks can be implemented directly in Python functions. For complex functions, a more complex sequence of steps may be necessary. We call a sequence of steps like this an **algorithm**.[6]

It is often useful to outline an algorithm schematically in a language we call **pseudocode**. This is a loose but programming-like language used to describe the algorithm without concern for syntax and implementation details. That is, pseudocode is used to express in structured natural language the semantics of a program without concern for its syntax in any specific programming language. The term "structured" here means some familiar programming structures—such as assignments, branches, loops, and functions—appear in pseudocode.

A **sorting algorithm** is an algorithm for sorting the elements of a list (e.g., numbers) by some metric (e.g., magnitude) such that the input list is returned ordered. There are many sorting algorithms with different efficiencies, but a relatively simple one is called **bubble sort**. For the sake of simplicity, we consider a list of n distinct numbers to be ordered such that they have increasing value. This algorithm repeatedly passes through the list, comparing adjacent elements and swapping their positions if they are out of order. After a single pass through the list, the greatest element will be in the last position because in every pairing, it is the greater. After the second pass, the second-greatest element will be in the penultimate position. After $n - 1$ passes, the list should be sorted.

In pseudocode, we can describe the algorithm more precisely, as shown in algorithm 1.

Algorithm 1 Pseudocode for the `bubble_sort_basic()` algorithm.

```
function bubble_sort_basic(list)
    for i ← 0, n − 1 do                                  ▷ Repeat n times
        for j ← 0, n − i − 1 do        ▷ Pass through potentially unsorted elements
            if list[j] > list[j + 1] then
                Swap list[j] and list[j + 1]
    return list
```

Can you think of a way to improve this algorithm? Often, when we write out the algorithm in pseudocode, it becomes more clear and improvements suggest themselves.

Once the algorithm for all complex functions are written in pseudocode, it is time to implement them as Python functions or methods. The functional analysis design

6. The term "algorithm" is actually quite broad, encompassing any technique for solving a problem.

diagrams of section 2.7.1 and the pseudocode algorithms from this section will help us rationalize this process and greatly improve our programs.

Box 2.3 Further Reading

- Abelson and Sussman (2016), a classic that teaches us how to think about computer programs
- Cross (2021), an engineering design methods (not specific to software) book with formal methods and useful case examples; see especially chapter 7 on the functional design method
- Hunt and Thomas (1999), a practical approach to designing programs, filled with nuggets of wisdom

2.8 Problems

Problem 2.1 WQ Write a program in a single script that meets the following requirements:

a. It imports the standard library `random` module.
b. It defines a function `rand_sub()` that defines a list of grammatical subjects (e.g., Jim, I, you, skeletons, a tiger, etc.) and returns a random subject; consider using `random.choice()` function.
c. It defines a function `rand_verb()` that defines a list of verbs in past tense (e.g., opened, smashed, ate, became, etc.) and returns a random verb.
d. It defines a function `rand_obj()` that defines a list of grammatical objects (e.g., the closet, her, crumbs, organs) and returns a random object.
e. It defines a function `rand_sen()` that returns a random subject-verb-object sentence as a string beginning with a capital letter and ending with a period.
f. It defines a function `rand_par()` that returns a random paragraph as a string composed of 3 to 5 sentences (the number of sentences should be random—consider using the `random.randint(a, b)` function that generates an `int` between `a` and `b`, inclusively). Sentences should be separated by a space `" "` character.
g. It calls `rand_par()` three times and prints the results.

Problem 2.2 SK Rewrite the program from problem 2.1 such that it meets the following requirements:

a. It defines the functions in a separate module with the file name `rand_speech_parts.py`.
b. Instead of defining the lists of subjects, verbs, and objects inside the functions, it assigns a variable to each list in the module's global namespace and accesses them from within the functions. Why is this preferable?
c. It imports the module into the main script.
d. It print three random paragraphs, as before.

Problem 2.3 YE Write a program in a single script that meets the following requirements:

a. It imports the standard library `random` module.
b. It defines a function `rand_step(x, d, ymax, wrap=True)` that returns a `float` that is the sum of `x` and a uniformly distributed random `float` between `-d` and `d`. Consider using the `random.uniform(a, b)` function that