

Author A. M. Kuchling

Release 0.32

In this document, we'll take a tour of Python's features suitable for implementing programs in a functional style. After an introduction to the concepts of functional programming, we'll look at language features such as iterators and generators and relevant library modules such as `itertools` and `functools`.

1 Introduction

This section explains the basic concept of functional programming; if you're just interested in learning about Python language features, skip to the next section on *Iterators*.

Programming languages support decomposing problems in several different ways:

- Most programming languages are **procedural**: programs are lists of instructions that tell the computer what to do with the program's input. C, Pascal, and even Unix shells are procedural languages.
- In **declarative** languages, you write a specification that describes the problem to be solved, and the language implementation figures out how to perform the computation efficiently. SQL is the declarative language you're most likely to be familiar with; a SQL query describes the data set you want to retrieve, and the SQL engine decides whether to scan tables or use indexes, which subclauses should be performed first, etc.
- **Object-oriented** programs manipulate collections of objects. Objects have internal state and support methods that query or modify this internal state in some way. Smalltalk and Java are object-oriented languages. C++ and Python are languages that support object-oriented programming, but don't force the use of object-oriented features.
- **Functional** programming decomposes a problem into a set of functions. Ideally, functions only take inputs and produce outputs, and don't have any internal state that affects the output produced for a given input. Well-known functional languages include the ML family (Standard ML, OCaml, and other variants) and Haskell.

The designers of some computer languages choose to emphasize one particular approach to programming. This often makes it difficult to write programs that use a different approach. Other languages are multi-paradigm languages that support several different approaches. Lisp, C++, and Python are multi-paradigm; you can write programs or libraries that are largely procedural, object-oriented, or functional in all of these languages. In a large program, different sections might be written using different approaches; the GUI might be object-oriented while the processing logic is procedural or functional, for example.

In a functional program, input flows through a set of functions. Each function operates on its input and produces some output. Functional style discourages functions with side effects that modify internal state or make other changes that aren't visible in the function's return value. Functions that have no side effects at all are called **purely functional**. Avoiding side effects means not using data structures that get updated as a program runs; every function's output must only depend on its input.

Some languages are very strict about purity and don't even have assignment statements such as `a=3` or `c = a + b`, but it's difficult to avoid all side effects. Printing to the screen or writing to a disk file are side effects, for example. For example, in Python a call to the `print()` or `time.sleep()` function both return no useful value; they're only called for their side effects of sending some text to the screen or pausing execution for a second.

Python programs written in functional style usually won't go to the extreme of avoiding all I/O or all assignments; instead, they'll provide a functional-appearing interface but will use non-functional features internally. For example, the implementation of a function will still use assignments to local variables, but won't modify global variables or have other side effects.

Functional programming can be considered the opposite of object-oriented programming. Objects are little capsules containing some internal state along with a collection of method calls that let you modify this state, and programs consist of making the right set of state changes. Functional programming wants to avoid state changes as much as possible and works with data flowing between functions. In Python you might combine the two approaches by writing functions that take and return instances representing objects in your application (e-mail messages, transactions, etc.).

Functional design may seem like an odd constraint to work under. Why should you avoid objects and side effects? There are theoretical and practical advantages to the functional style: