Functional Python Programming

Overview

Learn how to choose between imperative and functional approaches based on expressiveness, clarity, and performance

Get familiar with complex concepts such as monads, concurrency, and immutability

Apply functional Python to common Exploratory Data Analysis (EDA) programming problems

Objectives

Use Python's generator functions and generator expressions to work with collections in a non-strict (or lazy) manner

Utilize Python library modules including itertools, functools, multiprocessing, and concurrent features to ensure efficient functional programs

Use Python strings with object-oriented suffix notation and prefix notation

Avoid stateful classes with families of tuples

Design and implement decorators to create composite functions

Use functions such as max(), min(), map(), filter(), and sorted()

Write higher-order functions

Contents

Understanding Functional Programming

* Identifying a paradigm
* Subdividing the procedural paradigm
* A classic example of functional programming
* Exploratory data analysis

Introducing Essential Functional Concepts

* First-class functions
* Immutable data
* Strict and non-strict evaluation
* Recursion instead of an explicit loop state
* Functional type systems
* Familiar territory
* Learning some advanced concepts

Functions, Iterators, and Generators

* Writing pure functions
* Functions as first-class objects
* Using strings
* Using tuples and named tuples
* Cleaning raw data with generator functions
* Using lists, dicts, and sets

Working with Collections

* An overview of function varieties
* Working with iterables
* Using zip() to structure and flatten sequences
* Using reversed() to change the order
* Using enumerate() to include a sequence number

Higher-Order Functions

* Using max() and min() to find extrema
* Using Python lambda forms
* Lambdas and the lambda calculus
* Using the map() function to apply a function to a collection
* Using map() with multiple sequences
* Using the filter() function to pass or reject data
* Using filter() to identify outliers
* The iter() function with a sentinel value
* Using sorted() to put data in order
* Writing higher-order functions
* Writing higher-order mappings and filters
* Writing generator functions

Recursions and Reductions

* Simple numerical recursions
* Group-by reduction from many items to fewer

The Itertools Module

* Working with the infinite iterators
* Using the finite iterators
* Cloning iterators with tee()
* The itertools recipes

Decorator Design Techniques

* Decorators as higher-order functions
* Cross-cutting concerns
* Composite design
* Adding a parameter to a decorator

The Multiprocessing and Threading Modules

* Functional programming and concurrency
* What concurrency really means
* Using multiprocessing pools and tasks
* Using a multiprocessing pool for concurrent processing

Conditional Expressions and the Operator Module

* Evaluating conditional expressions
* Using the operator module instead of lambdas
* Starmapping with operators
* Reducing with operator module functions

The PyMonad Library