

Functional Programming TP

Costas Bouyioukos 2025

Overview

You'll build a scientific research analysis system using functional programming principles. The system will process scientist data, analyze research patterns, and generate reports while maintaining immutability and using higher-order functions.

Learning Objectives

- Master immutable data structures using `namedtuple` and `frozenset`
- Implement inner functions for encapsulation and closures
- Use lambda functions for simple transformations
- Apply map, filter, and reduce paradigms effectively

Setup

```
from collections import namedtuple
from functools import reduce
from typing import List, Tuple, Dict, Callable
import copy
from datetime import datetime

# Define immutable data structures
Scientist = namedtuple('Scientist', ['name', 'field', 'birth_year',
                                     'nobel_prize', 'nationality'])
Publication = namedtuple('Publication', ['title', 'author', 'year',
                                         'citations', 'journal'])
Research = namedtuple('Research', ['field', 'total_scientists',
                                   'avg_birth_year', 'nobel_winners', 'top_publications'])
Database = namedtuple('Database', ['scientists', 'publications',
                                   'research_areas'])

# Sample data to work with
sample_scientists = [
    Scientist('Marie Curie', 'Physics', 1867, 'Physics 1903, Chemistry 1911',
             'French'),
    Scientist('Albert Einstein', 'Physics', 1879, 'Physics 1921', 'German'),
    Scientist('Niels Bohr', 'Physics', 1885, 'Physics 1922', 'Danish'),
    Scientist('Linus Pauling', 'Chemistry', 1901, 'Chemistry 1954, Peace
1962', 'American'),
    Scientist('Barbara McClintock', 'Biology', 1902, 'Physiology/Medicine
1983', 'American'),
    Scientist('Dorothy Hodgkin', 'Chemistry', 1910, 'Chemistry 1964',
             'British'),
    Scientist('Rosalind Franklin', 'Chemistry', 1920, None, 'British'),
    Scientist('James Watson', 'Biology', 1928, 'Physiology/Medicine 1962',
```

```

'American'),
    Scientist('Francis Crick', 'Biology', 1916, 'Physiology/Medicine 1962',
'British'),
    Scientist('Chien-Shiung Wu', 'Physics', 1912, None, 'Chinese'),
    Scientist('Rita Levi-Montalcini', 'Biology', 1909, 'Physiology/Medicine
1986', 'Italian'),
    Scientist('Katherine Johnson', 'Mathematics', 1918, None, 'American'),
]

sample_publications = [
    Publication('On the Constitution of Atoms and Molecules', 'Niels Bohr',
1913, 2500, 'Philosophical Magazine'),
    Publication('The Structure of DNA', 'James Watson', 1953, 8000, 'Nature'),
    Publication('X-ray Studies of DNA', 'Rosalind Franklin', 1953, 1200,
'Acta Crystallographica'),
    Publication('The Nature of the Chemical Bond', 'Linus Pauling', 1939,
5000, 'Journal of American Chemical Society'),
    Publication('Radioactive Substances', 'Marie Curie', 1904, 3000, 'Annales
de Physique'),
    Publication('Genetic Control Systems', 'Barbara McClintock', 1961, 1500,
'Cold Spring Harbor Symposia'),
    Publication('Protein Crystallography', 'Dorothy Hodgkin', 1935, 2200,
'Nature'),
    Publication('Nerve Growth Factor', 'Rita Levi-Montalcini', 1960, 1800,
'Science'),
]

research_impact_scores = {
    'Physics': 9.2,
    'Chemistry': 8.8,
    'Biology': 9.0,
    'Mathematics': 8.5
}

```

Part 1: Basic Functional Operations

Exercise 1.1: Lambda Function Practice

```

def create_scientist_processors():
    """
    Return a dictionary of lambda functions for processing scientists:
    - 'is_nobel_winner': lambda that returns True if scientist has Nobel
prize
    - 'is_female': lambda that returns True if scientist is likely female
(basic name check)
    - 'field_prefix': lambda that returns first 4 letters of field
    - 'format_scientist': lambda that formats as "NAME (FIELD, BIRTH_YEAR)"
    """
    # TODO: Create and return dictionary of lambda functions
    # Hint: For female names, check if name starts with common female names
    female_names = {'Marie', 'Barbara', 'Dorothy', 'Rosalind', 'Chien-
Shiung', 'Rita', 'Katherine'}
    pass

```

Exercise 1.2: Scientist Filtering and Mapping

Write functions that use map, filter, and reduce to:

```
def analyze_basic_scientists(scientists: List[Scientist]) -> Dict:
    """
    Use map, filter, reduce to analyze scientists.
    Return a dictionary with:
    - 'nobel_winners': filtered scientists with Nobel prizes
    - 'total_birth_years': sum of all birth years
    - 'avg_birth_year': average birth year
    - 'scientist_names': list of all scientist names (uppercase)

    Use lambda functions where appropriate.
    """
    pass

def get_field_statistics(scientists: List[Scientist]) -> Dict[str, Dict]:
    """
    Create statistics per field using functional programming.
    Return dict where keys are fields and values are dicts with:
    - 'scientist_count': number of scientists in field
    - 'nobel_count': number of Nobel winners in field
    - 'avg_birth_year': average birth year in field
    - 'nobel_percentage': percentage who won Nobel prizes

    Use map, filter, reduce paradigm.
    """
    pass
```

Part 2: Inner Functions and Closures (45 minutes)

Exercise 2.1: Research Database Builder with Inner Functions

```
def create_research_analyzer(impact_threshold: float):
    """
    Create a research analyzer using inner functions and closures.
    Returns a function that can analyze scientists and publications.
    """

    def analyze_research_database(scientists: List[Scientist],
                                publications: List[Publication]) -> Database:
        """
        Inner function that analyzes scientists and publications to create
        database.
        Should categorize research and identify patterns.
        """

        def calculate_field_research(scientists: List[Scientist]) ->
List[Research]:
            """
            Nested inner function to calculate research metrics per field.
            Group scientists by field and calculate:
```

```

        - Total scientists in field
        - Average birth year
        - Number of Nobel winners
        - High-impact publications in field
        """
        # TODO: Implement research calculation
        pass

    def filter_high_impact_research(publications: List[Publication]) ->
    List[Publication]:
        """
        Nested inner function using the closure's impact_threshold.
        Filter publications that exceed the impact threshold.
        """
        # TODO: Use impact_threshold from closure
        pass

    # TODO: Use inner functions to build database
    pass

    return analyze_research_database

def create_scientist_filter_factory():
    """
    Create a factory function that returns customized filter functions.
    Uses closures to create specialized filters.
    """

    def create_era_filter(start_year: int, end_year: int):
        """Inner function that creates an era-based filter"""
        # TODO: Return a function that filters scientists by birth year range
        pass

    def create_nationality_filter(nationalities: frozenset):
        """Inner function that creates a nationality-based filter"""
        # TODO: Return a function that filters scientists by nationalities
        pass

    def create_field_group_filter(field_groups: Dict[str, frozenset]):
        """Inner function that creates field group filters"""
        # Example: {'STEM': frozenset(['Physics', 'Chemistry', 'Biology',
        'Mathematics'])}
        # TODO: Return a function that filters by field groups
        pass

    return {
        'era_filter': create_era_filter,
        'achievement_filter': create_achievement_filter,
        'nationality_filter': create_nationality_filter,
        'field_group_filter': create_field_group_filter
    }

```

Part 3: Advanced Map-Filter-Reduce (45 minutes)

Exercise 3.1: Immutable Scientific Report Generator

```
def create_scientific_report_generator():
    """
    Create a report generator that maintains immutability while
    building complex scientific analysis structures.
    """

    ScientificReport = namedtuple('ScientificReport', [
        'executive_summary', 'field_analysis', 'diversity_metrics',
        'collaboration_networks', 'recommendations', 'future_predictions'
    ])

    def generate_comprehensive_scientific_report(scientists: List[Scientist],
                                                publications:
List[Publication]) -> ScientificReport:
        """
        Generate a comprehensive scientific report using only immutable
        operations.
        No variables should be modified after creation.
        """

        def create_executive_summary(scientists: List[Scientist]) -> Dict:
            """Create executive summary using map-filter-reduce only"""
            # TODO: Create high-level statistics
            # Include: total scientists, fields covered, Nobel winners, time
            span, etc.
            pass

        def create_field_analysis(scientists: List[Scientist],
                                publications: List[Publication]) -> Dict[str,
Dict]:
            """Create detailed field analysis using functional methods"""
            # TODO: Analyze each field comprehensively
            # Include: scientist count, publication impact, Nobel rate, etc.
            pass

        def create_collaboration_networks(publications: List[Publication]) ->
Dict:
            """Create collaboration network analysis using pure functions"""
            # TODO: Map collaboration patterns
            pass

        def create_diversity_analysis(scientists: List[Scientist]) -> Dict:
            """Create diversity analysis using functional programming"""
            # TODO: Analyze representation across dimensions
            pass

        return generate_comprehensive_scientific_report

def create_scientific_ranking_system():
```

```

"""
Create a ranking system for scientists using functional programming.
"""

ScientistRanking = namedtuple('ScientistRanking', ['scientist', 'score',
'rank', 'category'])

def rank_scientists_by_impact(scientists: List[Scientist],
                             publications: List[Publication]) ->
List[ScientistRanking]:
    """
    Create scientist rankings using only functional operations.
    """

    def calculate_base_score(scientist: Scientist) -> float:
        """Calculate base impact score using functional approach"""
        # TODO: Score based on Nobel prizes, field impact, era, etc.
        pass

    def categorize_scientist(scientist: Scientist, score: float) -> str:
        """Categorize scientist based on score"""
        # TODO: Create categories like 'Legendary', 'Pioneering',
        'Influential', etc.
        pass

    # TODO: Combine all scoring functions to create final rankings
    pass

    return rank_scientists_by_impact

```

Bonus Challenges (If you finish early)








1. **Scientific Family Tree:** Create an immutable data structure to represent mentor-student relationships between scientists using functional programming.
2. **Citation Network Analysis:** Build a functional system to analyze citation networks between publications and identify influential papers.
3. **Nobel Prize Predictor:** Create a functional algorithm that predicts Nobel Prize likelihood based on scientist characteristics and publication patterns.
4. **Scientific Era Classifier:** Build a machine learning-style classifier using only functional programming to categorize scientists by scientific era based on their work patterns.

Key Concepts to Remember

1. **Immutable Data:** Once created, namedtuples and frozensets cannot be modified
2. **Inner Functions:** Functions defined inside other functions, useful for encapsulation
3. **Closures:** Inner functions that capture variables from outer scope
4. **Lambda Functions:** Anonymous functions for simple operations
5. **Map:** Transform each element in a collection

6. **Filter:** Select elements that meet certain criteria
7. **Reduce:** Combine all elements into a single result

Success Criteria

Your implementation should: -  Use only immutable data structures (no lists/dicts that get modified) -  Demonstrate proper use of inner functions and closures -  Include meaningful lambda functions (not just trivial examples) -  Use map, filter, and reduce appropriately throughout -  Maintain functional programming principles (no side effects) -  Handle the sample scientific data correctly and produce meaningful results -  Show understanding of scientific data relationships and patterns

Estimated Time: 2 hours - Part 1: 30 minutes - Part 2: 45 minutes
- Part 3: 45 minutes

Good luck! Focus on writing pure functions and maintaining immutability throughout your scientific data analysis implementation.