Python Programming

Unit 02 – Lecture 05: Advanced Functions and Functional Patterns

Tofik Ali

School of Computer Science, UPES Dehradun

February 14, 2026

Repository: `https://github.com/tali7c/Python-Programming`

Quick Links

Core Concepts Demo Interactive Summary

Agenda

Learning Outcomes

- Pass collections safely and reason about mutation

Learning Outcomes

- Pass collections safely and reason about mutation
- Apply recursion with a correct base case

Learning Outcomes

- Pass collections safely and reason about mutation
- Apply recursion with a correct base case
- Use `map`, `filter`, and `lambda`

Learning Outcomes

- Pass collections safely and reason about mutation
- Apply recursion with a correct base case
- Use `map`, `filter`, and `lambda`
- Recognize side effects in functions

Passing Collections

- Lists and dicts are mutable

```
def add_item(lst, item):
    lst.append(item)
```

Passing Collections

- Lists and dicts are mutable
- Functions can modify them in place

```
def add_item(lst, item):
    lst.append(item)
```

Passing Collections

- Lists and dicts are mutable
- Functions can modify them in place
- Use copies when you want to preserve input

```
def add_item(lst, item):
    lst.append(item)
```

Worked Example: Avoid Unintended Mutation

```
def sort_copy(items):
    tmp = items.copy()
    tmp.sort()
    return tmp

data = [3, 1, 2]
print(sort_copy(data))   # [1, 2, 3]
print(data)              # [3, 1, 2]
```

- Copy when you want a function without side effects

Recursion Basics

- A function calls itself

Recursion Basics

- A function calls itself
- Needs a base case to stop

Recursion Basics

- A function calls itself
- Needs a base case to stop
- Each call reduces the problem size

Recursive Example

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)
```

Pitfall: Recursion in Python

- Missing base case leads to infinite recursion

Pitfall: Recursion in Python

- Missing base case leads to infinite recursion
- Deep recursion can hit recursion limits

Pitfall: Recursion in Python

- Missing base case leads to infinite recursion
- Deep recursion can hit recursion limits
- Prefer iteration for simple loops in Python

map and filter

- map applies a function to each item

```
nums = [1, 2, 3, 4]
print(list(map(lambda x: x*x, nums)))
print(list(filter(lambda x: x%2==0, nums)))
```

map and filter

- `map` applies a function to each item
- `filter` keeps items that satisfy a condition

```
nums = [1, 2, 3, 4]
print(list(map(lambda x: x*x, nums)))
print(list(filter(lambda x: x%2==0, nums)))
```

Pitfall: map/filter Return Iterators

- In Python 3, `map` and `filter` return iterators

Pitfall: map/filter Return Iterators

- In Python 3, `map` and `filter` return iterators
- Wrap with `list(...)` to materialize results

Lambda and Inner Functions

- `lambda` creates small anonymous functions

```
def make_multiplier(k):
    def mult(x):
        return x * k
    return mult
```

Lambda and Inner Functions

- `lambda` creates small anonymous functions
- Inner functions can capture variables (closures)

```
def make_multiplier(k):
    def mult(x):
        return x * k
    return mult
```

Side Effects vs Pure Functions

- Pure functions do not change external state

Side Effects vs Pure Functions

- Pure functions do not change external state
- Side effects can make debugging harder

Side Effects vs Pure Functions

- Pure functions do not change external state
- Side effects can make debugging harder
- Prefer pure functions when possible

Demo: Cleaning Data with map/filter

- Remove invalid values

Script: `demo/functional_tools.py`

Demo: Cleaning Data with map/filter

- Remove invalid values
- Transform items with `map`

Script: `demo/functional_tools.py`

Demo: Cleaning Data with map/filter

- Remove invalid values
- Transform items with `map`
- Compute recursive sum

Script: `demo/functional_tools.py`

Demo: Cleaning Data with map/filter

- Remove invalid values
- Transform items with `map`
- Compute recursive sum
- Extension: rewrite using comprehensions and compare readability

Script: `demo/functional_tools.py`

Checkpoint 1

Identify the base case in a recursive function. Why is it necessary?

Checkpoint 2

Replace this loop with `map`:

```
nums = [1,2,3,4,5]
squares = []
for n in nums:
    squares.append(n*n)
```

Think-Pair-Share

When should you avoid recursion in Python? Provide one reason and one example.

Summary

- Passing collections can mutate inputs

Summary

- Passing collections can mutate inputs
- Recursion needs a base case

Summary

- Passing collections can mutate inputs
- Recursion needs a base case
- `map/filter/lambda` support functional style

Summary

- Passing collections can mutate inputs
- Recursion needs a base case
- `map/filter/lambda` support functional style
- Prefer pure functions for clarity and testing

Exit Question

If a function appends to a list passed as argument, does the caller see the change? Why?