
PCL2–Tutorial 05
Lucas Suomela, Rong Li, Tosca Peruzzi-Vieli, Isabelle Cretton

Welcome!

Functional Programming

Functional programming (FP) is a paradigm focused on writing software by composing pure functions, avoiding shared state, mutable data, and side-effects. It emphasizes the application of functions, in contrast to designing around data and bundling that data with its behaviour (OOP).

One of the most famous, purely functional programming languages is called Haskell, fun if you like math :)

In FP, you usually talk about:

- First-class functions
- High-order functions
- Pure functions

Deep Dive into Functions

Functions as First-Class Citizens:

They can be dynamically created, destroyed, passed to other functions, returned as values, and assigned to variables

→ Meaning you can use functions like any other data type!

Taking a function as an argument or returning a function allows for power abstractions.

An Example:

```
def greet(name): return f"Hello, {name}!"
say_hello = greet
print(say_hello("Alice"))
```

Embracing Pure Functions and Immutable Data

- Pure functions: No side effects (e.g., no modification of external variables), no reliance on external state, and idempotent (same inputs always result in the same output).
- Immutable data: Once created, these cannot be altered, leading to easier reasoning about the state of the program, and fewer bugs :)

Iterables, Iterators, Generators

Iterables

An object that can be looped over (iterated through) using a loop, like a for loop. An iterable implements the __iter__() method, which returns an iterator.

Iterator

An object that implements the iterator protocol, which consists of the methods __iter__() and __next__(). It allows to iterate over collections of objects (like lists or strings) one element at a time.

Generator

A special type of iterator defined using functions and the yield keyword. Unlike regular functions that return a single value and terminate, generators can yield multiple values, one at a time, pausing after each yield and resuming from there on the next call.

Anonymous Functions

- Also known as lambda functions.
- Used for creating simple, one-time-use functions without def.
- Defined with the lambda keyword.
- Can accept any number of arguments.
- Consist of only one expression.
- Automatically return the result of the expression.
- Ideal for temporary use, particularly with higher-order functions that require functions as arguments or as return values.

Example

```
books = [
    {'title': 'The Road', 'year': 2006},
    {'title': '1984', 'year': 1949},
    {'title': 'To Kill a Mockingbird', 'year': 1960},
    {'title': 'The Kite Runner', 'year': 2003}
]

recent_books = filter(lambda b: b['year'] > 2000, books)

print(list(recent_books))
```

Output

```
[{'title': 'The Road', 'year': 2006}, {'title': 'The Kite Runner', 'year': 2003}]
```

Decorators

Functions that modify the behaviour of another function without changing its code. They are a powerful tool for extending and modifying the behaviour of callable objects (functions, methods, and classes) without permanently modifying the callable itself.

Example

```
def count_calls(func):
    def inner(*args, **kwargs):
        inner.call_count += 1
        print(f"Call {inner.call_count} of {func.__name__}")
        return func(*args, **kwargs)
    inner.call_count = 0
    return inner

@count_calls
def say_hello():
    return "Hello!"

@count_calls
def say_goodbye():
    return "Goodbye!"

say_hello()
say_hello()
say_goodbye()
say_hello()
```

Output

```
Call 1 of say_hello
Call 2 of say_hello
Call 1 of say_goodbye
Call 3 of say_hello
```

In Python

For a demo in Python, you can run the Jupyter Notebook we have uploaded to OLAT:

PCL2_Tutorial_Week5_Demo.ipynb

Exercise 04

Objectives

Improve skills in functional programming by using:

- Pure functions
- Higher-order functions
- Anonymous functions
- Iterables

→ Whilst avoiding side effects

Deadline
March 26th at 23:59
(if you want feedback)

Reminder

You need to meet to meet these basic requirements for us to give feedback to / grade your submission:

- Make sure to submit before the deadline
- Add the tag "Submission"
- Make a release (after pushing your last changes to gitlab)

Feedback Exercise 02

Passing the pipeline is good…

but it doesn't replace thoroughly testing your code.

Doing the exercises last minute robs you of the possibility to challenge yourself (and deal with unexpected situations).

Focus on what you can learn from an exercise rather than just completing it. Use the exercises to expand your knowledge and skills.

Use all available resources – if they help you get better. Before asking ChatGPT, you might want to explore the Python documentation or Stack Overflow.

Your individual feedback is ready and is available on OLAT!

Now It's Your Turn...

To-Do

- ❑ Work on the exercise
- ❑ Issues with GitLab? Please let us know.
- ❑ Don't learn Haskell (unless you really want to?)
- ❑ ~~Enjoy your weekend!~~

Some "small" project ideas if you want to practice functional programming, they vary in difficulty: interactive Command-Line dictionary, a text processing tool using only built-in libraries, web scraper, stock market analyzer