Module 6: Programming Paradigms

Prof. Dr. Florian Haselbeck
Memoona Shakeel

Weihenstephan-Triesdorf University of Applied Sciences

Learning outcomes

Understand the three main styles of writing programs and recognize each one in simple Python code.

Key Topics

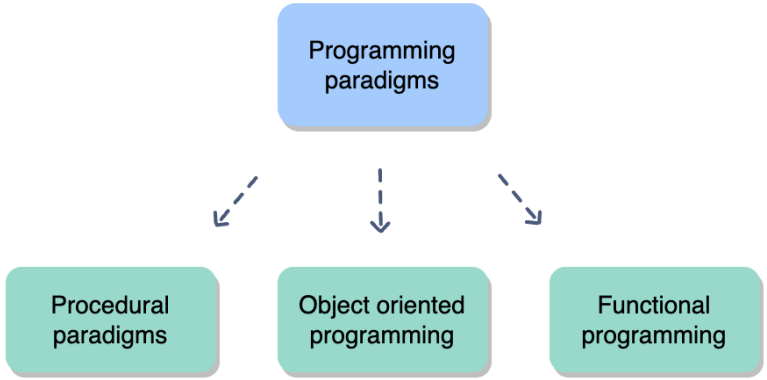
- What is a paradigm
- Procedural . OOP . Functional
- All three side by side
- Which one to use?

What is a programming paradigm?

Simple answer: A paradigm is just a style of writing code. There is no 'best' one. Each suits different jobs. We will focus on the three main paradigms in Python: Procedural, Object-Oriented (OOP), and Functional Programming.

A **programming language** is a tool used to solve problems. Different programmers may solve the same problem in different ways. These different styles of writing code are called programming paradigms.

For example, C mainly uses *Procedural Programming*, while C++ and Java use *Object-Oriented Programming (OOP)*. Python is flexible because it supports Procedural, OOP, and *Functional Programming*.



Programming paradigms available in Python

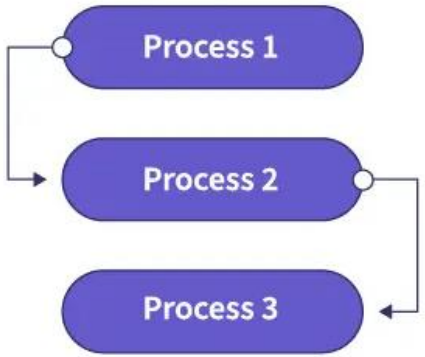
Procedural (step by step)

What it is: Write instructions in order. The program runs each line from top to bottom, like following a recipe. Code is structured hierarchically into blocks (such as if statements, loops, and functions). It is arguably the simplest form of coding. However, it can be difficult to write and maintain large and complex software due to its lack of enforced structure.

```
# Procedural Programming
# The program runs step by step

# Store soil moisture value
moisture = 25

# Check the moisture level
if moisture < 30:
    print("Soil is dry")
else:
    print("Soil moisture is good")
```



Object-Oriented Programming (OOP) classes and objects

What it is: Group related data and actions together. A class is the blueprint. An object is a real thing made from it.

Class:

Defines what an object HAS

and what it CAN DO.

Written once, used many times.

Object:

A real thing made from the class.

```
cow1 = Cow('Bessie', 24.7)
cow2 = Cow('Daisy', 21.2)
```

Good For:

Larger programs.

Many similar things: cows, fields, sensors.

Keeps code tidy.

Combines data and functions together.

```python
# Object-Oriented Programming (OOP)
# Using a class and object

# Create a class
class Sensor:

    # Create a method to check soil
    def check_soil(self, moisture):

        # Check moisture level
        if moisture < 30:
            print("Soil is dry")
        else:
            print("Soil moisture is good")

# Create an object from the class
sensor = Sensor()

# Call the method
sensor.check_soil(25)
```

Functional (small functions, data in → result out)

What it is: Write small functions that each do one clear job. Pass data in, get a result back. Think of it as a pipeline. A function does ONE job. Same input = same output. No changing things outside.

Looks Like:

```
def add(a, b):
    return a + b
```

Small, focused functions.

List comprehensions.

Good for:

Processing lists of data.

Filtering sensor readings.

Math calculations.

Data pipelines.

```
# Functional Programming
# Using a function to return a result

# Create a function
def check_soil(moisture):

    # Return result based on condition
    return "Soil is dry" if moisture < 30 else "Soil moisture is good"

# Call the function and print result
print(check_soil(25))
```

Same task three different styles

Procedural

def, if, for

```python
# Store milk yield values
yields = [24.7, 21.2, 19.8]

# Start total at 0
total = 0

# Add each value one by one
for y in yields:
    total = total + y

# Calculate average
avg = total / len(yields)

# Print result
print(avg)    # 21.9
```

Object Oriented

Class, self

```python
# Create a class
class Herd:

    # Store data inside object
    def __init__(self, data):
        self.yields = data

    # Method to calculate average
    def average(self):
        return sum(self.yields) / len(self.yields)

# Create object
herd = Herd([24.7, 21.2, 19.8])

# Print result
print(herd.average())    # 21.9
```

Functional

def, return

```python
# Store milk yield values
yields = [24.7, 21.2, 19.8]

# Function to calculate average
def average(data):
    return sum(data) / len(data)

# Print result
print(average(yields))    # 21.9
```

Which style should I use?

Short answer: Start procedural. Move to OOP when your program has many similar things. Use functional ideas everywhere **for data tasks.**

Writing a short sensor-reading script?
→ **Procedural**

Managing 50 fields, each with name + crop?
→ **OOP**

Filtering or transforming a list of values?
→ **Functional**

A big farming app with data + reports?
→ **Mix all three**

Overview of key content

- **Procedural** = write instructions in order, step by step
  Use variables, if/else, for loops, and functions called one after another.
  This is the style you already write, and the best place to start.

- **OOP** = group data and actions into a class · an object is one real instance
  class Cow: is the blueprint. cow1 = Cow('Bessie', 24.7) is the real thing.
  Each object has its own data. Good for programs with many similar things.

- **Functional** = small functions, data in → result out
  Each function does one job and returns a value.
  def average(data): return sum(data)/len(data) clear input, clear output, easy to test.

- **All three can be mixed**, use what makes your code clearest
  Short scripts: procedural. Many similar objects: OOP. Data filtering: functional.
  Python lets you use all three freely in the same project.

Thank you!

Contact Detail

Prof. Dr. Florian Haselbeck

florian.haselbeck@hswt.de

Raum D1.230a

For any questions, please feel free to use the Moodle-Course Forum!