Féidearthachtaí as Cuimse
Infinite Possibilities

Transition to Object Oriented Programming– Colette Kirwan

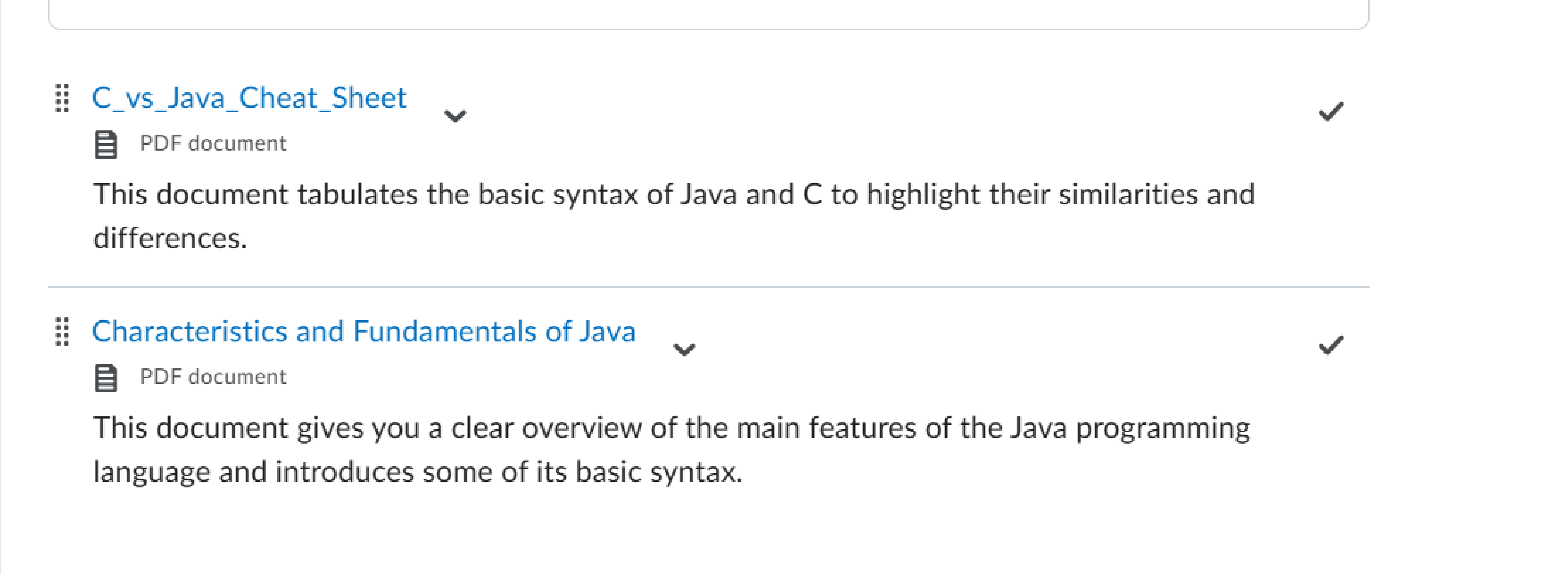
Week 1 2025, Object-Oriented Programming

OLLSCOIL TEICNEOLAÍOCHTA
BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL
UNIVERSITY DUBLIN

Overview

- Change from C to Java
- Programming Paradigms

Documents to read in the Toolkit folder



C_vs_Java_Cheat_Sheet

PDF document

This document tabulates the basic syntax of Java and C to highlight their similarities and differences.

Characteristics and Fundamentals of Java

PDF document

This document gives you a clear overview of the main features of the Java programming language and introduces some of its basic syntax.

Overview

> “A **programming paradigm** is a **way or style of writing code**. It’s a set of **rules and concepts** that guide how programs are structured and executed

Introduction to Programming Paradigms

Procedural (and Imperative) → C, Python (when using functions), bash scripts

Describes step-by-step instructions

Object-Oriented (OOP) → Java, Kotlin, Python

Organizes code around objects and classes.

Functional (FP) → Python, Kotlin, R, Java Streams, Lisp

Focuses on pure functions and immutable data

Declarative → SQL, HTML, Bash (crontab)

Focuses on what to do not how to do it

Hybrid (Mix of Paradigms) → Java, Python, Kotlin

These support multiple paradigms (OOP, functional, procedural)

Overview (Imperative)

Imperative:

- Code is executed step by step in a sequential manner.
- Uses loops and conditionals,.
- Focuses on *how* tasks are performed.
- In English – imperative means give commands – in programming you tell computer how to do something- sequence of commands

Procedural Programming (subset of imperative) is a **structured way of writing imperative code** using **functions (procedures)** to organize and reuse logic.

✓ Procedural Programming is always Imperative.
✓ Imperative Programming is NOT always Procedural (because it doesn't need functions).

...

Example in Scripts

Bash scripting is a mix of:

1. **Programming logic** (loops, conditionals, variables, functions).

2. **Linux commands** (used for file manipulation, system tasks).

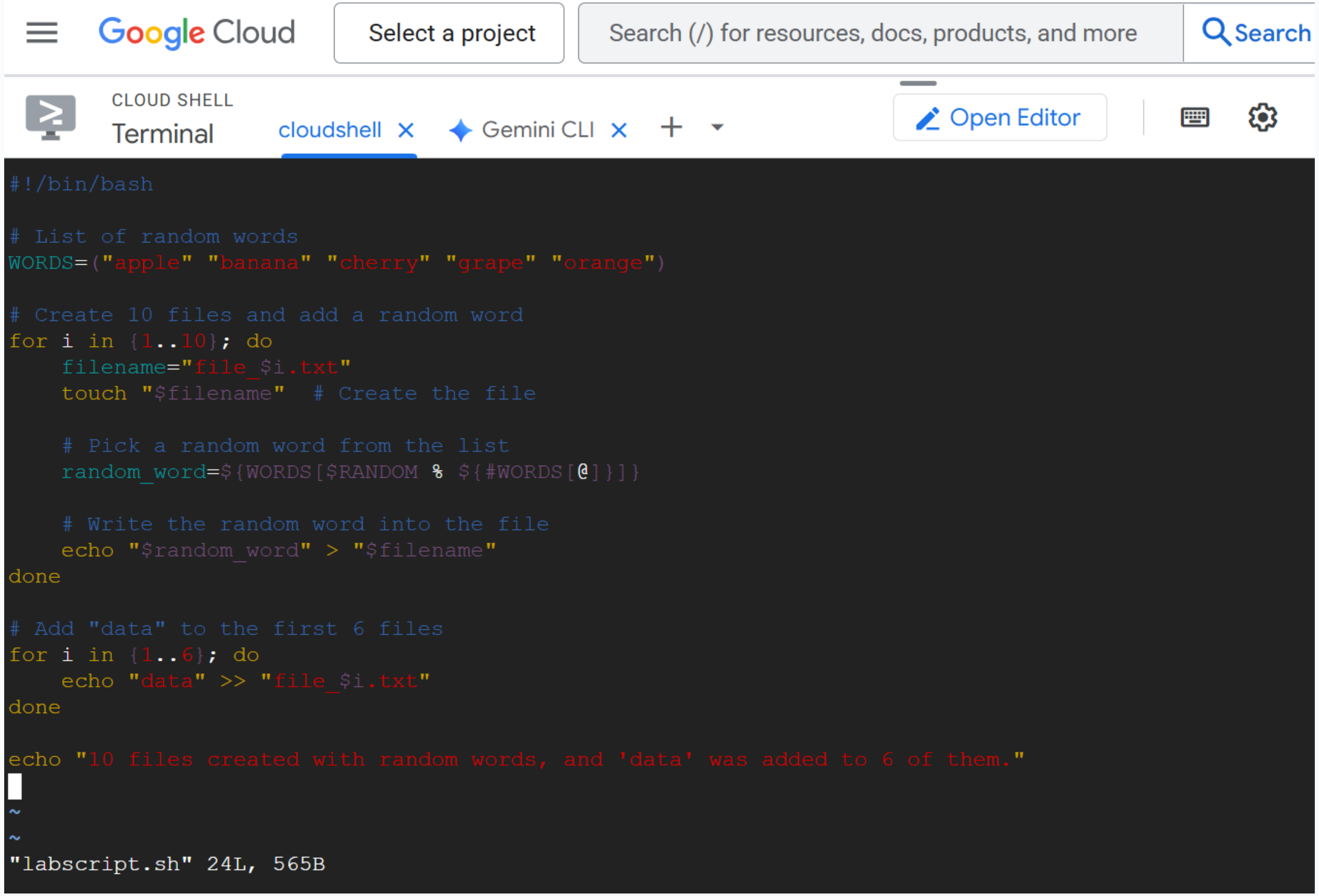
3. ✅ Control Structures → if, for, while, case

   ✅ Linux Commands → ls, grep, echo, pwd, rm, etc.

   ✅ Functions → my_function() { }

4. Bash is both a scripting language and a shell that runs Linux commands!

Class Demo



```
#!/bin/bash

# List of random words
WORDS=("apple" "banana" "cherry" "grape" "orange")

# Create 10 files and add a random word
for i in {1..10}; do
    filename="file_$i.txt"
    touch "$filename"  # Create the file

    # Pick a random word from the list
    random_word=${WORDS[$RANDOM % ${#WORDS[@]}]}

    # Write the random word into the file
    echo "$random_word" > "$filename"
done

# Add "data" to the first 6 files
for i in {1..6}; do
    echo "data" >> "file_$i.txt"
done

echo "10 files created with random words, and 'data' was added to 6 of them."

~
~
"labscript.sh" 24L, 565B
```

Procedural Programming (Imperative)

Overview

- Based on the concept of **procedures** (also called functions, routines, or subroutines). It follows a **step-by-step** approach to solving problems by breaking them down into sequences of instructions.
- Code is executed step by step in a sequential manner.
- Uses loops, conditionals, and functions.
- Focuses on *how* tasks are performed.

Procedural Programming (Imperative)

Linear Execution Flow – The program is executed in a logical order, from top to bottom, unless controlled by loops or function calls.

Procedures (Functions) – Code is organized into reusable blocks (functions) that perform specific tasks.

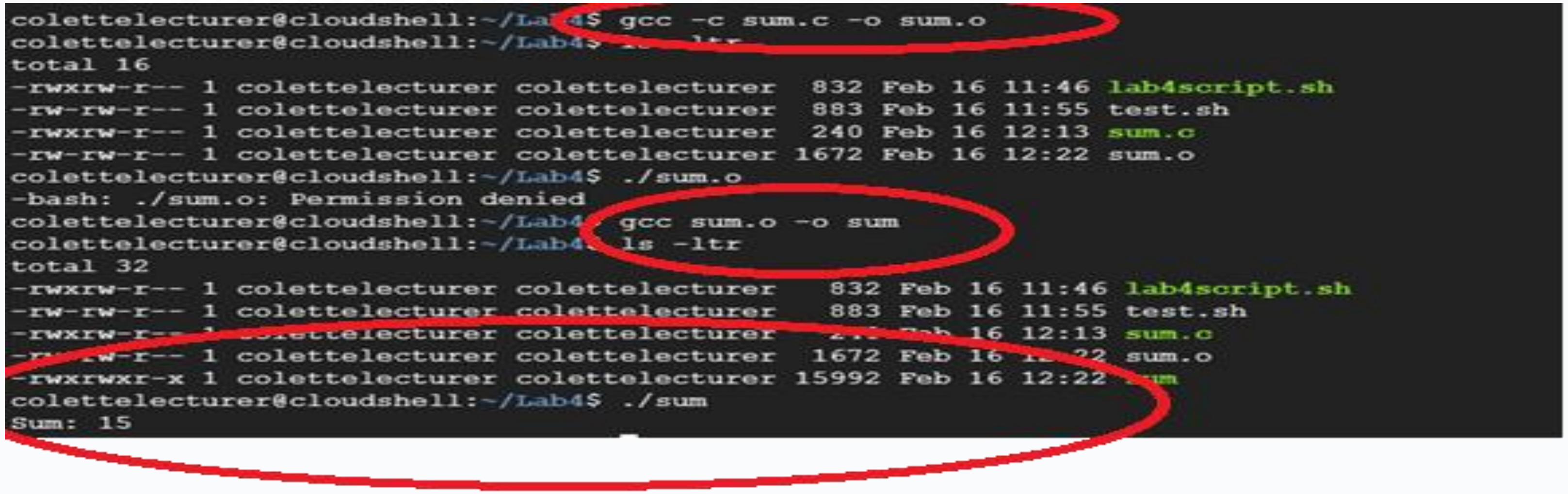
Variables and Data Structures – Data is stored in variables, arrays, and other structures, and it is manipulated within functions.

Control Structures – It uses loops (for, while), conditionals (if, else), and function calls to control the flow of execution.

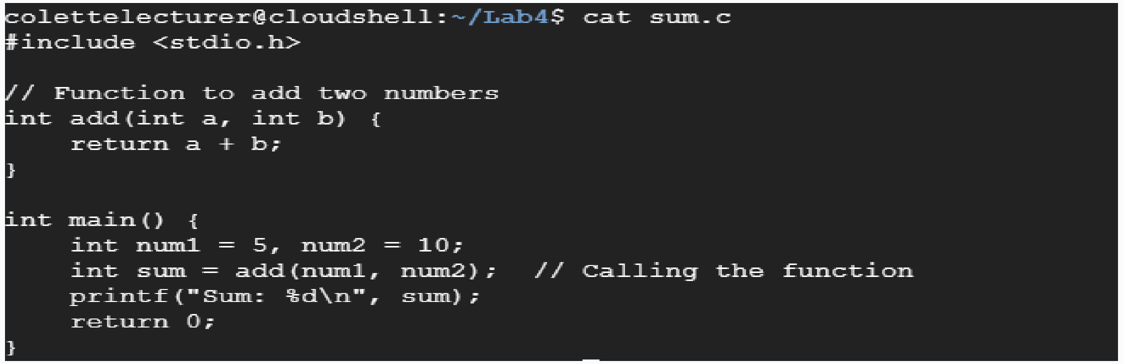
Global and Local Variables – Variables can be defined within functions (local) or outside functions (global).

Modular Approach – Code is divided into smaller functions to improve readability, maintainability, and reusability.

Compile the C file : Class DEMO



```
colettelecturer@cloudshell:~/Lab4$ gcc -c sum.c -o sum.o
colettelecturer@cloudshell:~/Lab4$ ls -ltr
total 16
-rwxrw-r-- 1 colettelecturer colettelecturer  832 Feb 16 11:46 lab4script.sh
-rw-rw-r-- 1 colettelecturer colettelecturer  883 Feb 16 11:55 test.sh
-rwxrw-r-- 1 colettelecturer colettelecturer  240 Feb 16 12:13 sum.c
-rw-rw-r-- 1 colettelecturer colettelecturer 1672 Feb 16 12:22 sum.o
colettelecturer@cloudshell:~/Lab4$ ./sum.o
-bash: ./sum.o: Permission denied
colettelecturer@cloudshell:~/Lab4$ gcc sum.o -o sum
colettelecturer@cloudshell:~/Lab4$ ls -ltr
total 32
-rwxrw-r-- 1 colettelecturer colettelecturer   832 Feb 16 11:46 lab4script.sh
-rw-rw-r-- 1 colettelecturer colettelecturer   883 Feb 16 11:55 test.sh
-rwxrw-r-- 1 colettelecturer colettelecturer   240 Feb 16 12:13 sum.c
-rw-rw-r-- 1 colettelecturer colettelecturer  1672 Feb 16 12:22 sum.o
-rwxrwxr-x 1 colettelecturer colettelecturer 15992 Feb 16 12:22 sum
colettelecturer@cloudshell:~/Lab4$ ./sum
Sum: 15
```



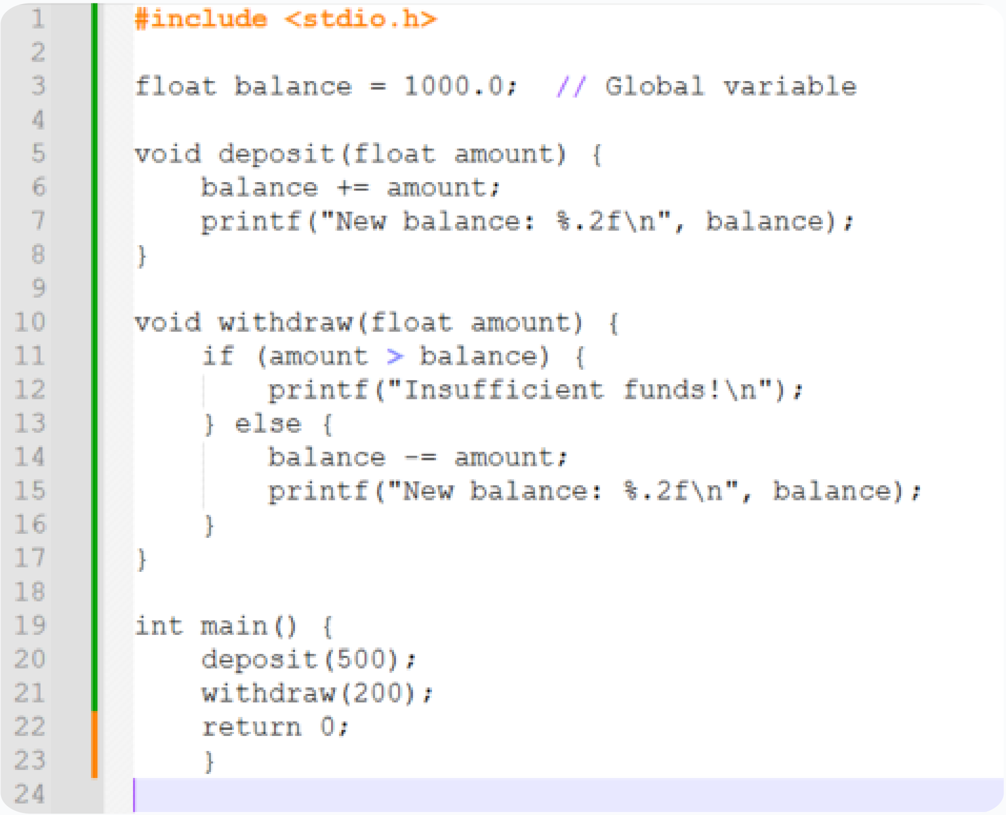
```
colettelecturer@cloudshell:~/Lab4$ cat sum.c
#include <stdio.h>

// Function to add two numbers
int add(int a, int b) {
    return a + b;
}

int main() {
    int num1 = 5, num2 = 10;
    int sum = add(num1, num2);  // Calling the function
    printf("Sum: %d\n", sum);
    return 0;
}
```

Advantages of Procedural Programming

- Simple and easy to understand.
- Efficient for small projects. (Great for automation, simple applications and quick prototyping)
- Easier debugging due to clear flow control.

Advantage	Explanation	Example Languages
Simple & Easy to Understand	Follows **step-by-step** execution, using functions for modularity. (top-down structure and functions break down complex tasks into smaller manageable parts). Simple and direct – great for system level programming like Operating Systems	C, Python, Bash
Efficient for Small Projects	No complex structures needed (no need for objects, classes for e.g. C), ideal for scripts and utilities	Bash, C, Pascal
Easier Debugging	Linear execution, modular testing, and fewer interactions. (written in a linear step-by Step manner – easier to develop, read and modify)	Python, C

Explanation of disadvantages



```c
#include <stdio.h>

float balance = 1000.0;   // Global variable

void deposit(float amount) {
    balance += amount;
    printf("New balance: %.2f\n", balance);
}

void withdraw(float amount) {
    if (amount > balance) {
        printf("Insufficient funds!\n");
    } else {
        balance -= amount;
        printf("New balance: %.2f\n", balance);
    }
}

int main() {
    deposit(500);
    withdraw(200);
    return 0;
    }
```

- issues when **scaling:**
- The global variable balance is accessible everywhere, leading to uncontrolled modifications.
- **More functions = more complexity** → Harder to track dependencies.
- If we want to add user accounts, we need to rewrite the whole logic.

- **Code Duplication in Procedural Programming**

- If we need two types of transactions (for **savings** and **checking** accounts), we must **write two similar function**

Disadvantages of Procedural Programming

- **Difficult to scale** for large applications.
- **Lacks reusability** compared to OOP.
- **Code duplication** is common.

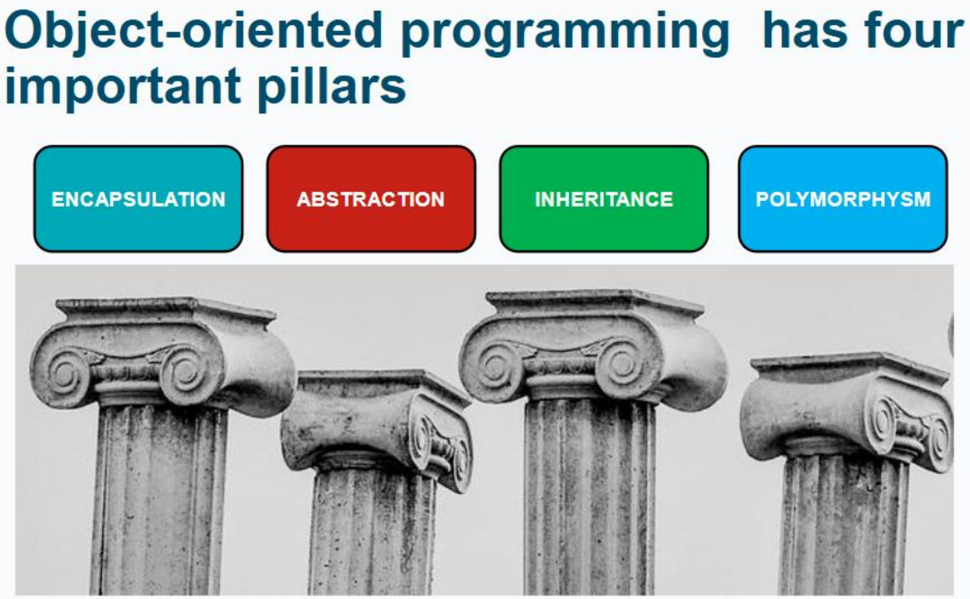
Background

- **Imperative programming is the most basic programming style**—it tells the computer exactly **how** to perform tasks. **It gives full control but can become complex without structure**.
- **Procedural Programming** evolved to **improve imperative programming**.
- OOP evolved **to address the limitations** of procedural programming, especially **code reusability and maintainability**, but it didn't directly replace procedural programming. Procedural and OOP are **both widely used today**.

.

Object-Oriented Programming (OOP)

- **Object-oriented programming (OOP)** – A programming paradigm based on the representation of a program as a set of objects and interactions between them
- Organizes code using objects and classes.
- Encapsulation, Inheritance, abstraction, Polymorphism.
- Models real-world entities.



Object-oriented programming has four important pillars

Introduction to Object Oriented Programming

- Not a language! Many languages use OOP.
- A programming paradigm for organizing and structuring code.
- Shifts from focusing on procedures to managing objects.
- Provides a more modular and organized approach to programming.
- Essential for creating complex and scalable software applications.

Transition from Procedural to OOP

- In procedural programming, code is organized around procedures or functions.
- Code can become complex and hard to maintain as it grows.
- Object-Oriented Programming (OOP) introduces a new way of structuring code. It really is a new way of thinking.
- OOP focuses on creating and managing objects that encapsulate both data and behaviour.
- Objects represent real-world entities and their interactions.
- OOP promotes reusability, modularity and better organization of code.

Understanding the Paradigm Shift

- OOP shifts from **procedure**-centred to **object**-centred approach.
- Code is organized around objects with attributes and methods.
- Objects mimic real-world entities and their interactions.
- Encourages breaking down complex problems into manageable components.
- Promotes better code **organization**, **reusability**, and **maintainability**.
- **Paradigm shifts requires a shift in mindset and coding approach**.

Key Concepts: Classes and Objects

- **Classes**: Blueprint or template for creating objects.
- **Objects**: Instances of classes with attributes and methods.
- **Attributes**: Data or characteristics associated with an object.
- **Methods**: Functions defined in classes to perform actions.
- **Encapsulation**: Bundling data and methods into a single unit (object).
- **Abstraction**: Hiding complex implementation details, focusing on essential features.
- **Inheritance**: Creating new classes based on existing ones, inheriting attributes and methods.
- **Polymorphism**: Ability to use different classes through a common interface.