

Exercise walk-through

20.1

Programming Paradigms

A-Level Computer Science

A paradigm is a style of programming

Four paradigms sit in this syllabus.

low-level

machine code or assembly; registers and addresses; drivers, firmware

imperative

a sequence of commands that change state – Topics 9 and 11

object-oriented

objects bundle data and methods; instances of classes

declarative

say *what*, not *how* – SQL, Prolog, Haskell

Programming paradigms

Imperative programming says *how*; **declarative programming** says *what*, and splits into **functional programming** and **logic programming**.

low-level
close to the hardware

imperative
a sequence of commands

object-oriented
objects with data + methods

declarative
say what, not how

Low-level and imperative

Low-level programming – **assembly language** working directly with **memory addresses** – is close to the hardware. Each instruction maps to what the CPU runs. Addressing modes: immediate, direct, indirect, indexed, relative. Fast and compact; architecture-specific and hard to maintain.

Imperative (procedural / structured): assignments, conditionals, loops, function calls. **Variables** hold state; statements change it. Strong when the algorithm has clear sequential steps. Python, C.

Declarative – what, not how

Two kinds:

- **functional functions effects** – **pure** with no **side effects** (Haskell, Lisp)
- **logic** – facts and rules; the engine answers a goal (Prolog)

SQL is the familiar example: `SELECT * FROM Customer WHERE Country = 'UK'` says what you want, not how to walk the records.

Modern languages **mix** paradigms. Python supports procedural, OOP and functional. Pick the one that fits the problem.

The paradigms, and what each is for

low-level	maximum control and speed – assembly
imperative	a sequence of statements changing state – most general-purpose languages
object-oriented	objects holding data and behaviour together – large systems, GUIs, simulations and games
declarative	state <i>what</i> , not <i>how</i> – logic programming
database	data queries – SQL
the point	a paradigm is a style , not a language: many languages support several

Choose by the problem's shape. A query belongs in SQL, a simulation in objects, a device driver in something low-level – and the mark is for saying why.

Exercises 2.1-2.3

2.1 State what a **programming paradigm** is.

[1]

2.2 Name the **four** features (pillars) of **OOP**.

[2]

2.3 State what **encapsulation** means.

[1]

Exercises 2.1-2.3

2.1 State what a **programming paradigm** is. [1]

2.1

- a **style/approach to structuring programs**, with its own concepts and language features

2.2 Name the **four** features (pillars) of **OOP**. [2]

2.2

- **encapsulation, inheritance, polymorphism, abstraction**

2.3 State what **encapsulation** means. [1]

2.3

- an object's **data is hidden** and accessed **only through its methods** (not directly)

Exercises 2.4·3.1

2.4 Give **one** example of a **declarative** language.

[1]

3.1 Describe what is meant by the OOP feature **inheritance**, with an **example**.

[3]

Exercises 2.4-3.1

2.4 Give **one** example of a **declarative** language.

[1]

2.4

- any one: **SQL**, **Prolog**, **Haskell**, **Lisp**, **F#**

3.1 Describe what is meant by the OOP feature **inheritance**, with an **example**.

[3]

3.1

- a **subclass** inherits the **attributes and methods** of a **superclass**
- and can add or **override** its own
- example: a **Manager** inherits from **Employee** ("is-a")

Exercises 3.2-3.3

3.2 Identify the OOP feature that **restricts external access to an object's data**, and state **one** benefit of it. [2]

3.3 Explain what is meant by **polymorphism**. [2]

Exercises 3.2-3.3

3.2 Identify the OOP feature that **restricts external access to an object's data**, and state **one** benefit of it. [2]

3.2

- **encapsulation**
- benefit: it **protects the data** from invalid changes / lets the internals change without breaking other code

3.3 Explain what is meant by **polymorphism**. [2]

3.3

- different object types respond to the **same method call** in their **own way**
- so the caller need not know the exact type (e.g. `Circle` and `Rectangle` each implement `Area()`)

Exercise 3.4

3.4 State **one** characteristic of **low-level** programming and **one** of **declarative** programming. [2]

Exercise 3.4

3.4 State **one** characteristic of **low-level** programming and **one** of **declarative** programming. [2]

3.4

- low-level: any one – close to hardware, direct register/memory access, fast/compact, architecture-specific
- declarative: any one – you state **what** not **how**, no explicit step-by-step control flow

Exercise 3.5

3.5 A class `Book` has a private attribute `Title` of type `STRING`. Write pseudocode for a **constructor** that sets `Title`, and a **get method** `GetTitle()` that returns it. [4]

Exercise 3.5

3.5 A class `Book` has a private attribute `Title` of type `STRING`. Write pseudocode for a **constructor** that sets `Title`, and a **get method** `GetTitle()` that returns it. [4]

3.5

- constructor `NEW` with a parameter; getter returns `Title`

```
PUBLIC PROCEDURE NEW(GivenTitle : STRING)
    Title ← GivenTitle
ENDPROCEDURE
```

```
PUBLIC FUNCTION GetTitle() RETURNS STRING
    RETURN Title
ENDFUNCTION
```