Unit 1a: Programming Paradigms

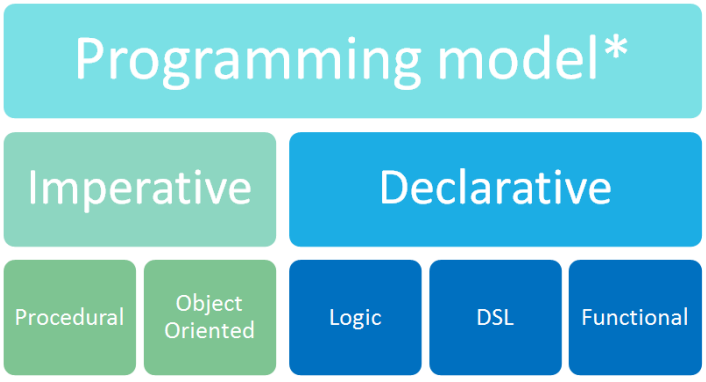
1 Introduction

The first part of the module involves reflecting on different approaches to thinking about a problem solution. Traditionally, an imperative approach has been popular as it reflects the underlying Von Neumann architecture of the majority of processors.

Towards the end of the unit we will contrast this approach with a functional approach by examining the theories of computation of Alan Turing and Alonso Church.

2 Programming Paradigm

The *programming paradigm*, or *model*, determines how the (human) programmer expresses ideas to the computer in order to get the computer to do useful computation. Programming creates a solution to a specified problem within a specific problem domain; different programming paradigms are suited to different kinds of computation that may be required to create a solution or to different problem domains, or, may be suited to the way in which the programmer formulates their ideas. It can be very difficult for programmers who have learned one model well to adapt to another model. Paradigms are not always mutually exclusive or opposite to each other, many languages are hybrids to some extent and support aspects of more than one programming paradigm.



3 Imperative Approach

An imperative solution uses a sequence of *statements* to determine how to reach a certain goal; these statements are said to *change the state* of the program as each one is executed in turn. The focus is on how to perform tasks (algorithms) and how to track changes in state.

Algorithms are expressed with statements and control structures (if statements, loops) and program counter and variables track state.

Here is a simple imperative-style code example (C#, but typical of many languages) – think about the order of execution, what variable values change and why:

```
int sum = 0;
foreach (int i in mylist) {
   sum += (i + 1);
}
```

OO *and* procedural languages usually support imperative programming. OO or procedural programming models determine what a method or procedure should do, and imperative code inside methods and procedures implements the task.

Imperative languages are generally *Turing complete* i.e. this essentially means they can implement any algorithm that can be designed. Turing completeness relates to idea of the Turing machine, proposed by computer scientist and Enigma codebreaker Alan Turing – we will look at this later.

Imperative programming contrasts with *declarative* programming.

3 Declarative Approach

Declarative programming expresses the logic of a computation without describing its control flow. It may attempt to minimize or eliminate side effects by describing what the program must accomplish in terms of the problem domain, rather than describe how to accomplish it as a sequence of the programming language primitives e.g. SQL describes the data required but not the steps required to get the data from a database:

```
SELECT firstname, lastname FROM People
WHERE age > 18
```

This requires a database engine to execute the query which may decide for itself the best way to implement this.

Some, though certainly not all, declarative languages are not Turing complete. (See TIOBE note concerning SQL).

4 Summary of programming models

Note this is by no means an exhaustive listing of programming models!

4.1 Popularity of programming models

None of the key models listed here are new by any means. Functional languages have existed since the 1950s, object-oriented since the 1970s. The popularity of each has risen and/or fallen, depending on which types of applications programmers have typically been creating.

Object-oriented emerged as the dominant programming methodology from the mid 1990s when programming languages supporting the model became widely available.

Functional programming has become increasingly popular recently, due to its suitability for applications in concurrent and parallel programming and in finance and data science applications, the availability of powerful and flexible new languages (such as Scala) and increasing support for functional approaches in mainstream languages.

We will look in more detail at examples of languages which support the models described here, focussing on functional in particular.