Next Generation Higher National Unit Specification

Programming Paradigms (SCQF level 8)

Unit code: J7DY 48

SCQF level: 8 (16 SCQF credit points)

Valid from: session 2023–24

Prototype unit specification for use in pilot delivery only (version 1.0) June 2023

This unit specification provides detailed information about the unit to ensure consistent and transparent assessment year on year.

This unit specification is for teachers and lecturers and contains all the mandatory information required to deliver and assess the unit.

This edition: June 2023 (version 1.0)

Unit purpose

This is a specialist unit, designed for learners with an interest in computer science who wish to learn about programming paradigms and the programming languages and techniques associated with each paradigm. It is particularly relevant to learners who wish to progress to higher levels of study in computing.

Before starting the unit, learners should have previous programming experience and be confident in using at least one programming language. Learners will find it helpful to have completed, or to do concurrently with the unit, Object-oriented Programming at SCQF level 8 or Algorithms and Data Structures at SCQF level 8.

During the unit, learners cover:

- the historical development of programming languages and how they can be categorised into paradigms
- the main characteristics of each paradigm
- comparing paradigms
- applying different paradigms to a problem

The unit helps learners to understand the different types of programming paradigm and the best applications of each. It broadens their appreciation of the art of computer programming.

The unit is knowledge-based but learners apply their theoretical knowledge of paradigms to the solution of a problem, using at least two paradigms. The focus of the unit is breadth, not depth. The unit aims to expose learners to a range of alternative paradigms, and their associated programming techniques and use-cases, to give them an appreciation of the different languages and different techniques that can be applied to problems.

On completion of the unit, learners understand that there are different types of programming language, and different programming techniques associated with each type. They appreciate how different paradigms can be used to solve problems. Learners could progress to further studies in computer science at SCQF level 9 and above.

Unit outcomes

Learners who complete this unit can:

1 describe the evolution of programming languages and programming paradigms
2 explain programming paradigms and their relationship to programming languages
3 compare programming paradigms
4 select and apply paradigms to a problem

Evidence requirements

Learners must provide knowledge and product evidence for this unit.

Knowledge evidence

Knowledge evidence relates to outcomes 1, 2 and 3. Evidence may be the minimum to infer competence. Learners must demonstrate all the defined knowledge at least once.

In the context of outcome 1, learners must describe all the important milestones.

In the context of outcome 2, learners must explain imperative and declarative paradigms, and provide several examples of each.

In the context of outcome 3, learners must compare the following paradigms (or instances of paradigms):

- low-level languages
- procedural languages
- object-oriented languages
- event-driven languages
- functional languages
- logic languages

They must consider the key characteristics, advantages and disadvantages, and use-cases of each paradigm.

If you use a test, you can sample knowledge evidence. The sampling frame must always include question(s) from outcomes 1, 2 and 3.

Learners must produce knowledge evidence under controlled conditions in terms of location, timing and supervision.

Product evidence

Product evidence relates to outcome 4. Learners must solve a problem using at least two of the paradigms compared in outcome 3. They must provide a narrative explaining the differences between the paradigms and the strengths and weaknesses of each in the context of the problem. The problem must be sufficiently complex to demonstrate the key characteristics of each approach.

Learners should produce product evidence in lightly-controlled conditions, over an extended period of time, at times and places at their discretion. Learners must produce evidence without assistance. However, they may access code repositories where appropriate.
The standard of evidence should be consistent with the SCQF level of the unit.

Evidence produced in lightly-controlled conditions must be authenticated. The Guide to Assessment provides further advice on methods of authentication. The 'Additional guidance' section in this unit specification provides specific examples of instruments of assessment that will generate the required evidence.

Knowledge and skills

The following table shows the knowledge and skills covered by the unit outcomes:

Knowledge	Skills
Learners should understand: ♦ historical milestones in the evolution of programming languages and paradigms ♦ contemporary programming languages and their uses ♦ trends in programming ♦ fundamental components of programming languages, including: — constants — variables — sub-routines — functions — data structures ♦ machine code and low-level languages ♦ high-level languages ♦ program translation, including assemblers and compilers ♦ syntax and semantics ♦ algorithms and pseudocode ♦ programming constructs ♦ concept of paradigms in the context of programming ♦ types of programming paradigm ♦ characteristics of each paradigm ♦ classification of languages by paradigm ♦ multi-paradigms ♦ difference between procedural programming and object-oriented programming (OOP) ♦ programming techniques related to each paradigm ♦ strengths and weaknesses of each paradigm ♦ performance of each paradigm ♦ applications of each paradigm	Learners can: ♦ select a paradigm to solve a problem ♦ apply an appropriate paradigm to a problem ♦ write code using an imperative programming language ♦ write code using a declarative programming language

Meta-skills

Throughout this unit, learners develop meta-skills to enhance their employability in the computing sector.

Self-management

This meta-skill includes:

- focusing: sorting; attention to detail
- adapting: decision making; selecting and applying paradigms in a range of contexts

Social intelligence

This meta-skill includes:

- communicating: receiving information through various means and interpreting it
- collaborating: listening and conveying information

Innovation

This meta-skill includes:

- curiosity: recognising problems and devising solutions
- creativity: idea generation; maker mentality
- sense-making: pattern recognition; analysis; synthesis
- critical thinking: logical and computational thinking; deconstruction; judgment

Delivery of unit

You can deliver this unit alongside Object-oriented Programming at SCQF level 8 or Algorithms and Data Structures at SCQF level 8. The time required varies depending on the previous experience of individual learners.

Based on 80 hours delivery and assessment time, we suggest the following distribution:

Outcome 1 — Describe the evolution of programming languages and programming paradigms
(15 hours)

Outcome 2 — Explain programming paradigms and their relationship to programming languages
(25 hours)

Outcome 3 — Compare programming paradigms
(15 hours)

Outcome 4 — Select and apply paradigms to a problem
(25 hours)

Additional guidance

The guidance in this section is not mandatory.

Content and context for this unit

You can offer this as a stand-alone unit or as part of a group award, such as HND Computer Science. When taken as part of a larger qualification, you can deliver it alongside other units in the award, such as Algorithms and Data Structures at SCQF level 8 or Object-oriented Programming at SCQF level 8 in HND Computer Science.

The unit is particularly suitable for learners who wish to progress to an undergraduate degree in computer science as it aligns with content found in most university computer science curriculums in years one and two.

Before starting the unit, learners should be proficient in at least one high-level (most likely to be imperative) programming language. Time constraints limit the depth of treatment of each language and paradigm. The key objective of the unit is to give learners an appreciation of different languages and different programming techniques, and their typical use cases. For example, if learners come into the unit with prior experience of Python, they could highlight and contrast the multi-paradigm nature of that language (object-oriented and procedural) with other examples of each paradigm, such as Pascal (procedural) and Visual Basic (OOP).

Further exposure to languages such as Prolog (logic) and Structured Query Language (SQL) (functional) would illustrate all the main paradigms. Highlighting the recent changes to Excel, which made it a fully-fledged functional programming language, would consolidate many of the key concepts in the unit.

The unit provides learners with an opportunity to explore the historical development of programming languages and programming paradigms. It broadens their appreciation of computer programming.

Although you should use traditional classifications of paradigms (imperative and declarative) for the purposes of learning and teaching, the following types of language could be considered distinct paradigms in terms of their language constructs and semantics, and their associated methods:

- low-level languages
- procedural languages
- object-oriented languages
- event-driven languages
- functional languages
- logic languages

Learners are not required to use all these languages and methods. However, they must use at least one imperative and one declarative language. Learners should be exposed to more than one of each type.

You should consider the performance of each paradigm (knowledge) in relation to the following characteristic of a language or paradigm:

- instruction path length
- code complexity
- execution and response times
- object code size
- memory management
- parallel programming

Writing code (skills) may be limited to straightforward applications of each paradigm. The focus of the unit is breadth, not depth.

Resources

Learners must have access to a range of programming languages, and their corresponding integrated development environments (IDEs), to carry out the practical element of the course. We suggest the following languages:

Procedural

- C: developed by Dennis Ritchie and Ken Thompson
- C++: developed by Bjarne Stroustrup
- ColdFusion: developed by J J Allaire
- Java: developed by James Gosling at Sun Microsystems
- Pascal: developed by Niklaus Wirth

Object-oriented

- C++: developed by Bjarne Stroustrup
- Java: developed by James Gosling at Sun Microsystems
- Objective-C: designed by Brad Cox
- Visual Basic .NET: developed by Microsoft
- Python: developed by Guido van Rossum
- Ruby: developed by Yukihiro Matsumoto
- Simula: developed by Ole-Johan Dahl and Kristen Nygaard
- Smalltalk: developed by Alan Kay, Dan Ingalls and Adele Goldberg

Logic

- Prolog: developed by Alain Colmerauer and Robert Kowalski

Functional

- Erlang: developed by Joe Armstrong and Robert Virding
- Excel: developed by Microsoft
- JavaScript: developed by Brendan Eich
- Haskell: developed by Lennart Augustsson and Dave Barton
- Lisp: developed by John McCarthy
- ML: developed by Robin Milner
- Scala: developed by Martin Odersky
- SQL: developed by IBM Corporation, Inc.

Approaches to delivery

You can teach the unit alongside other units within a group award.

When teaching the unit, you should aim to embed theory concepts in the practical work of the course. Learners should experience a range of programming languages and associated programming methods. Brief exposure to a variety of paradigms is preferable to deep exposure to a small number of paradigms, even if exposure is trivial in terms of problem complexity.

Approaches to assessment

A traditional approach to assessment might involve a test of learners' knowledge of outcomes 1, 2 and 3 and a practical assignment for outcome 4.

The test could take the form of a timed, unseen question paper comprising several extended-response questions. Sample questions could include:

- Describe THREE important milestones in the development of programming languages, explaining their contribution to the evolution of computer programming.
- Explain FOUR significant differences between imperative and declarative paradigms, providing examples to illustrate your answer.
- What is meant by 'multi-paradigm'? Give at least ONE example of a multi-paradigm programming language and explain the benefits of this type of language.

A pass mark of 50 per cent is appropriate.

An alternative traditional approach could require learners to write one or more essays on specific themes within the unit. If you use this assessment method, the essay topic(s) must be unknown to learners and the assessment carried out in timed and supervised conditions.

The programming assignment could ask learners to solve a problem using two different paradigms (such as procedural and OOPs). The assessment activity would require learners to provide program code (for each approach) and a narrative outlining the differences between the paradigms, and the strengths and weaknesses of each in the context of the

problem. If you use this assessment method, it should be carried out in lightly-controlled conditions over an extended period of time.

A more contemporary approach to assessment could involve learners maintaining a portfolio of their written work (knowledge evidence) and programming activities (product evidence). If you take this approach, the portfolio must satisfy the specific evidence requirements defined in the ‘Evidence requirements’ section; sampling is not appropriate.

Equality and inclusion

This unit is designed to be as fair and as accessible as possible with no unnecessary barriers to learning or assessment.

You should take into account the needs of individual learners when planning learning experiences, selecting assessment methods or considering alternative evidence.

Guidance on assessment arrangements for disabled learners and/or those with additional support needs is available on the assessment arrangements web page: www.sqa.org.uk/assessmentarrangements.

Information for learners

Programming Paradigms (SCQF level 8)

This information explains:

- what the unit is about
- what you should know or be able to do before you start
- what you need to do during the unit
- opportunities for further learning and employment

Unit information

This unit introduces you to different programming languages and programming methods. You should be familiar with at least one programming language before starting the unit.

The unit is particularly suitable for learners who wish to progress to study degree-level computer science.

During the unit, you learn about:

- the historical development of programming languages
- programming paradigms
- different programming techniques
- machine code programming
- high-level languages
- object-oriented languages
- functional programming languages
- logic programming languages
- when and how to use different paradigms

You gain hands-on experience of using a range of programming languages and programming methods.

The unit deepens your appreciation of programming languages and programming techniques.

Assessment could take the form of a test of your knowledge, and a practical assignment to demonstrate that you can program using two or more different types of programming languages.

Throughout the unit, you develop meta-skills covering self-management, social intelligence and innovation.
On completion of the unit, you may choose to progress to units of study in computer science at SCQF level 9 and above.

Administrative information

Published: June 2023 (version 1.0)

Superclass: CB

History of changes

Version	Description of change	Date

Note: please check SQA's website to ensure you are using the most up-to-date version of this document.