Outcomes and content for Year 11

Programming fundamentals

Outcomes

A student:

- describes methods used to plan, develop and engineer software solutions **SE-11-01**
- explains how structural elements are used to develop programming code **SE-11-02**
- describes how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-11-03**
- applies safe and secure practices to collect, use and store data **SE-11-04**
- applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
- implements safe and secure programming solutions **SE-11-07**

Content

Software development

- Explore fundamental software development steps used by programmers when designing software

  Including:
  - requirements definition
  - determining specifications
  - design
  - development
  - integration
  - testing and debugging
  - installation
  - maintenance
- Research and evaluate the prevalence and use of online code collaboration tools

Designing algorithms

- Apply computational thinking and algorithmic design by defining the key features of standard algorithms, including sequence, selection, iteration and identifying data that should be stored
- Apply divide and conquer and backtracking as algorithmic design strategies
- Develop structured algorithms using pseudocode and flowcharts, including the use of subprograms
- Use modelling tools including structure charts, abstraction and refinement diagrams to support top-down and bottom-up design
- Analyse the logic and structure of written algorithms

  Including:
  - determining inputs and outputs
  - determining the purpose of the algorithm
  - desk checking and peer checking
  - determining connections of written algorithms to other subroutines or functions
- Identify procedures and functions in an algorithm
- Experiment with object-oriented programming, imperative, logic and functional programming paradigms

Data for software engineering

- Investigate the use of number systems for computing purposes, including binary, decimal and hexadecimal
- Represent integers using two's complement
- Investigate standard data types

  Including:
  - char (character) and string
  - Boolean
  - real
  - single precision floating point
  - integer
  - date and time
- Create data dictionaries as a tool to describe data and data types, structure data, and record relationships
- Use data structures of arrays, records, trees and sequential files

Developing solutions with code

- Apply skills in computational thinking and programming to develop a software solution

  Including:
  - converting an algorithm into code
  - using control structures
  - using data structures
  - using standard modules
  - creating relevant subprograms that incorporate parameter passing
- Implement data structures that support data storage

  Including:
  - single and multidimensional arrays
  - lists
  - trees
  - stacks
  - hash tables
- Compare the execution of the Waterfall and Agile project management models as applied to software development
- Test and evaluate solutions, considering key aspects including functionality, performance, readability of code, quality of documentation
- Use debugging tools

  Including:
  - breakpoints
  - single line stepping
  - watches
  - interfaces between functions
  - debugging output statements
  - debugging software available in an integrated development environment (IDE)

- Determine sets of suitable test data

  Including:
  - boundary values
  - path coverage
  - faulty and abnormal data
- Determine typical errors experienced when developing code, including syntax, logic and runtime, and explain their likely causes

The object-oriented paradigm

Outcomes

A student:

- describes methods used to plan, develop and engineer software solutions **SE-11-01**
- explains how structural elements are used to develop programming code **SE-11-02**
- describes how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-11-03**
- applies safe and secure practices to collect, use and store data **SE-11-04**
- applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
- implements safe and secure programming solutions **SE-11-07**
- applies language structures to refine code **SE-11-08**
- manages and documents the development of a software project **SE-11-09**

Content

Understanding OOP

- Apply the key features of an object-oriented programming (OOP) language

  Including:
  - objects
  - classes
  - encapsulation
  - abstraction
  - inheritance
  - generalisation
  - polymorphism
- Compare procedural programming with OOP
- Use data flow diagrams, structure charts and class diagrams to represent a system
- Describe the process of design used to develop code in an OOP language

  Including:
  - task definition
  - top-down and bottom-up
  - facade pattern
  - agility
- Assess the effectiveness of programming code developed to implement an algorithm
- Investigate how OOP languages handle message-passing between objects
- Explain code optimisation in software engineering
- Outline the features of OOP that support collaborative code development

  Including:
  - consistency
  - code commenting
  - version control
  - feedback

Programming in OOP

- Design and implement computer programs involving branching, iteration and functions in an OOP language for an identified need or opportunity

- Implement and modify OOP programming code

  Including:
  - clear and uncluttered mainline
  - one logical task per subroutine
  - use of stubs
  - use of control structures and data structures
  - ease of maintenance
  - version control
  - regular backup

- Apply methodologies to test and evaluate code

  Including:
  - unit, subsystem and system testing
  - black, white and grey box testing
  - quality assurance

Programming mechatronics

Outcomes

A student:

- describes methods used to plan, develop and engineer software solutions **SE-11-01**
- explains how structural elements are used to develop programming code **SE-11-02**
- describes how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-11-03**
- applies safe and secure practices to collect, use and store data **SE-11-04**
- describes the social, ethical and legal implications of software engineering on the individual, society and the environment **SE-11-05**
- applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
- implements safe and secure programming solutions **SE-11-07**
- applies language structures to refine code **SE-11-08**
- manages and documents the development of a software project **SE-11-09**

Content

Understanding mechatronic hardware and software

- Outline applications of mechatronic systems in a variety of specialised fields
- Identify the hardware requirements to run a program and the effect on code development

  Including:
  - assessing the relationship of microcontrollers and the central processing unit (CPU)
  - the influence of instruction set and opcodes
  - the use of address and data registers

- Identify and describe a range of sensors, actuators and end effectors/manipulators within existing mechatronic systems

  Including:
  - motion sensors
  - light level sensors
  - hydraulic actuators
  - robotic grippers

- Use different types of data and understand how it is obtained and processed in a mechatronic system, including diagnostic data and data used for optimisation
- Experiment with software to control interactions and dependencies within mechatronic systems

  Including:
  - motion constraints
  - degrees of freedom
  - combination of subsystems
  - combination of sensors, actuators and end effectors to create viable subsystems

- Determine power, battery and material requirements for components of a mechatronic system
- Develop wiring diagrams for a mechatronic system, considering data and power supply requirements
- Determine specialist requirements that influence the design and functions of mechatronic systems designed for people with disability

Designing control algorithms

- Develop, modify and apply algorithms to control a mechatronic system
- Explore the algorithmic patterns, code and applications for open and closed control systems
- Outline the features of an algorithm and program code used for autonomous control

Programming and building

- Design, develop and produce a mechatronic system for a real-world problem

  Including:
  - software control
  - mechanical engineering
  - electronics and mathematics
- Implement algorithms and design programming code to drive mechatronic devices
- Develop simulations and prototypes of a potential mechatronic system to test programming code
- Design, develop and implement programming code for a closed loop control system
- Apply programming code to integrate sensors, actuators and end effectors/manipulators
- Implement specific control algorithms that enhance the performance of a mechatronic system
- Design, develop and implement a user interface (UI) to control a mechatronic system
- Create and use unit tests to determine the effectiveness and repeatability of each component's control algorithm