NSW Education Standards Authority

Higher School Certificate Course Specifications

Software Engineering

NESA acknowledges Traditional Owners and Custodians of Country throughout NSW, and pays respect to Elders past and present. NESA recognises Aboriginal Peoples' continuing Cultures and Connections to lands, waters, skies and Community.

© 2023 NSW Education Standards Authority

The documents on the NSW Education Standards Authority (NESA) website and the NSW Curriculum website contain material prepared by NESA for and on behalf of the Crown in right of the State of New South Wales. The material is protected by Crown copyright.

These websites hold the only official and up-to-date versions of the documents available on the internet. Any other copies of these documents, or parts of these documents, that may be found elsewhere on the internet might not be current and are not authorised. You cannot rely on copies from any other source.

All rights are reserved. No part of the material may be:

- reproduced in Australia or in any other country by any process, electronic or otherwise, in any material form
- transmitted to any other person or stored electronically in any form without the written permission of NESA except as permitted by the *Copyright Act 1968* (Cth).

When you access the material, you agree:

- to use the material for research or study, criticism or review, reporting news and parody or satire
- to use the material for information purposes only
- not to modify the material or any part of the material without the written permission of NESA
- to reproduce a single copy for personal bona fide study use only and not to reproduce any major extract or the entire material without the permission of NESA
- to include this copyright notice in any copy made
- to acknowledge that NESA is the source of the material.

The documents may include third-party copyright material such as photos, diagrams, quotations, cartoons and artworks. This material is protected by Australian and international copyright laws and may not be reproduced or transmitted in any format without the copyright owner's permission. Unauthorised reproduction, transmission or commercial use of such copyright material may result in prosecution.

NESA has made all reasonable attempts to locate the owners of third-party copyright material. NESA invites anyone from whom permission has not been sought to contact the Copyright Officer.

Special arrangements applying to the NSW Curriculum Reform

As part of the NSW Curriculum Reform process, NESA grants a limited non-exclusive licence to:

- teachers employed in NSW government schools and registered non-government schools
- parents of children registered for home schooling

to use, modify and adapt the NSW syllabuses for **non-commercial educational use** only. The adaptation must not have the effect of bringing NESA into disrepute.

Note: The above arrangements do not apply to private/home tutoring companies, professional learning service providers, publishers, and other organisations.

For more information on the above or for **commercial use or any other purpose**, please contact the Copyright Officer for permission.

Email: copyright@nesa.nsw.edu.au

D2022/521381

Table of Contents

System and Data Modelling Tools 5

Data flow diagrams 5

Structure charts 7

Data dictionary 8

Class diagrams 9

Storyboard 9

Decision trees 10

Project Management Tools 11

Gantt charts 11

Process diaries / log books 12

Programming Paradigms 13

Object-oriented paradigm 13

Logic paradigm 13

Imperative paradigm 13

Functional paradigm 13

Algorithms 14

Pseudocode 14

Flowcharts 14

Control Structures 15

Sequence 15

Selection 15

Binary selection 15

Multi-way selection 16

Nested IF 16

Repetition. 17

Pre-test 17

Post-test 17

FOR / NEXT 18

Subroutines 19

Using a subroutine with one parameter 19

Using a subroutine with multiple parameters 20

Passing a value back from a function 21

Relational Databases 22

SQL 22

Object-Relational Mapping (ORM) 23

Wiring diagrams for mechatronic systems 24

Programming for the Web 25

Front-end web development frameworks 25

Cross-site scripting 25

Cascading Style Sheets (CSS) 26

Machine Learning 27

Machine learning automation through DevOps 27

Regression algorithms 28

Neural Networks 29

Training cycle 29

Execution cycle. 29

Methods for Testing a System 30

Character Representation 31

Programming with Python 32

Introduction

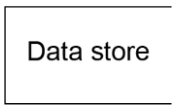
Software Engineering Course Specifications are an integral part of the course content for Year 11 and Year 12 and indicate the depth of study required for some concepts in the *Software Engineering 11–12 Syllabus*. The *Software Engineering 11–12 Syllabus* must be applied in conjunction with the Software Engineering Course Specifications.

System and Data Modelling Tools

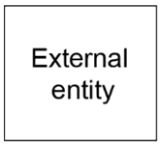
Data flow diagrams

Symbols

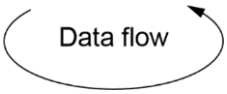
A circle represents a process. A process uses input(s) to generate output(s).



A data store can be an electronic file or non-computer storage.

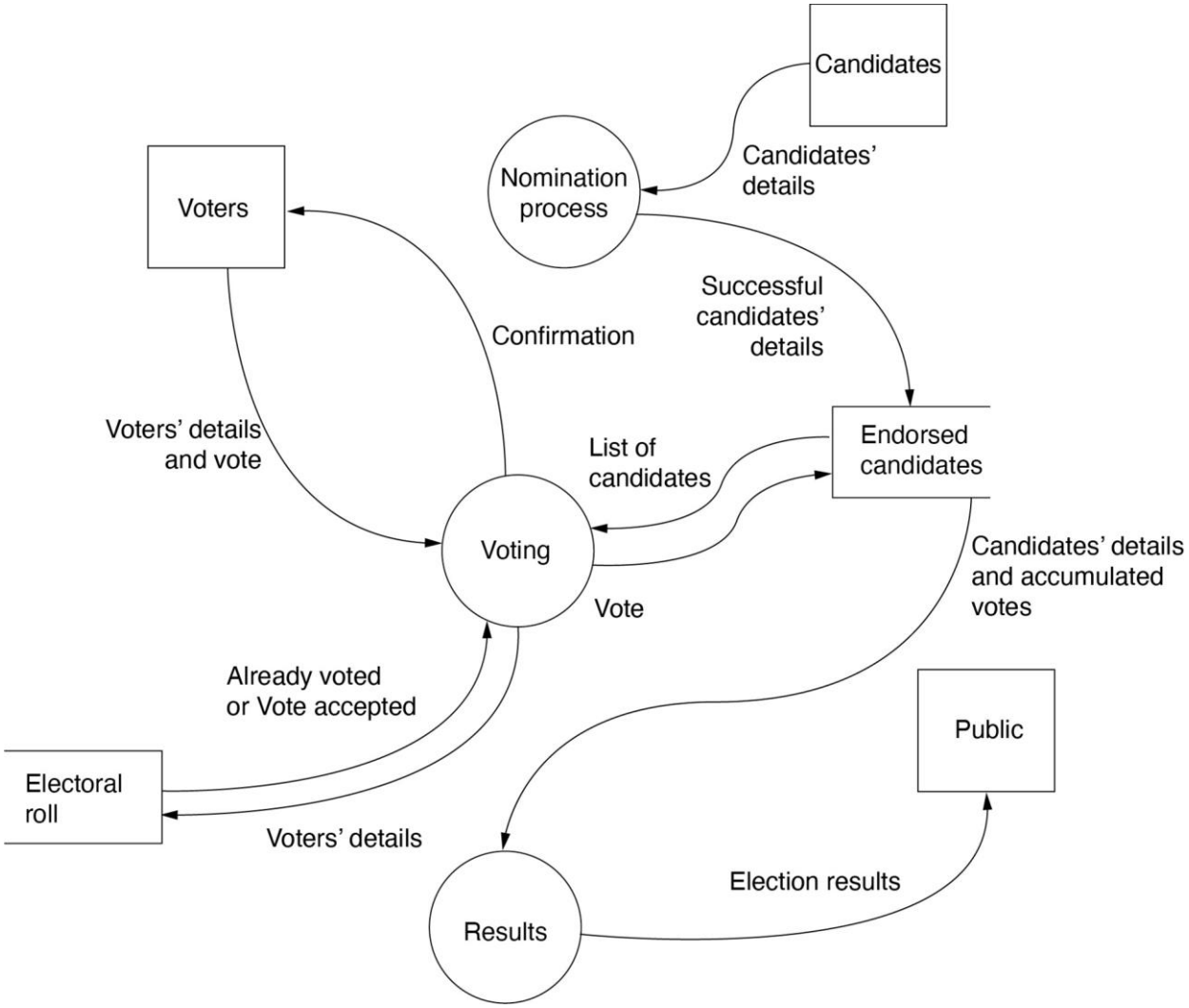


An external entity can be any person, organisation or element that provides data to the system or receives data from the system.



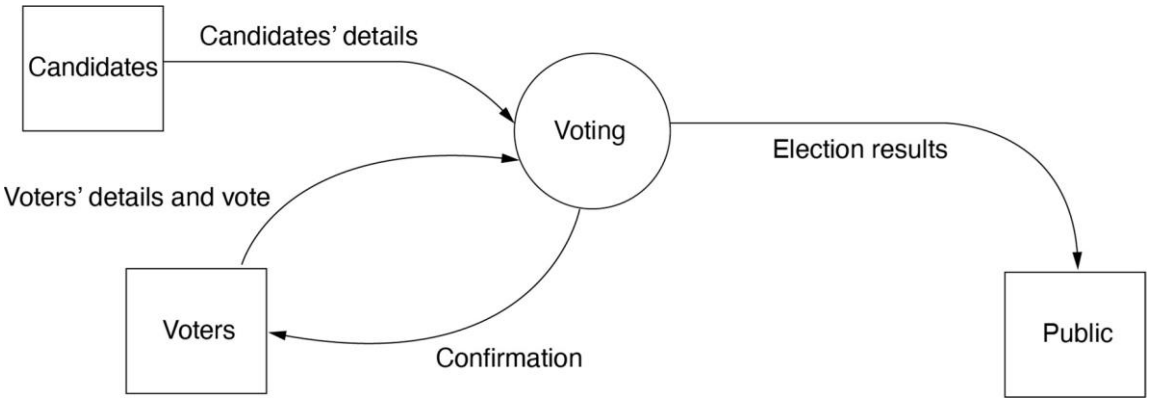
A labelled, curved arrow represents the flow of data between processes, data stores and external entities.

The following data flow diagram models a voting system.



Level 0 data flow diagram

Level 0 data flow diagrams represent an overview of the entire system and do not show data stores or internal processes. The following represents a Level 0 data flow diagram for the voting system.



Structure charts

Structure charts represent a system by showing the separate subroutines that make up the system and their relationship to each other.

Symbols

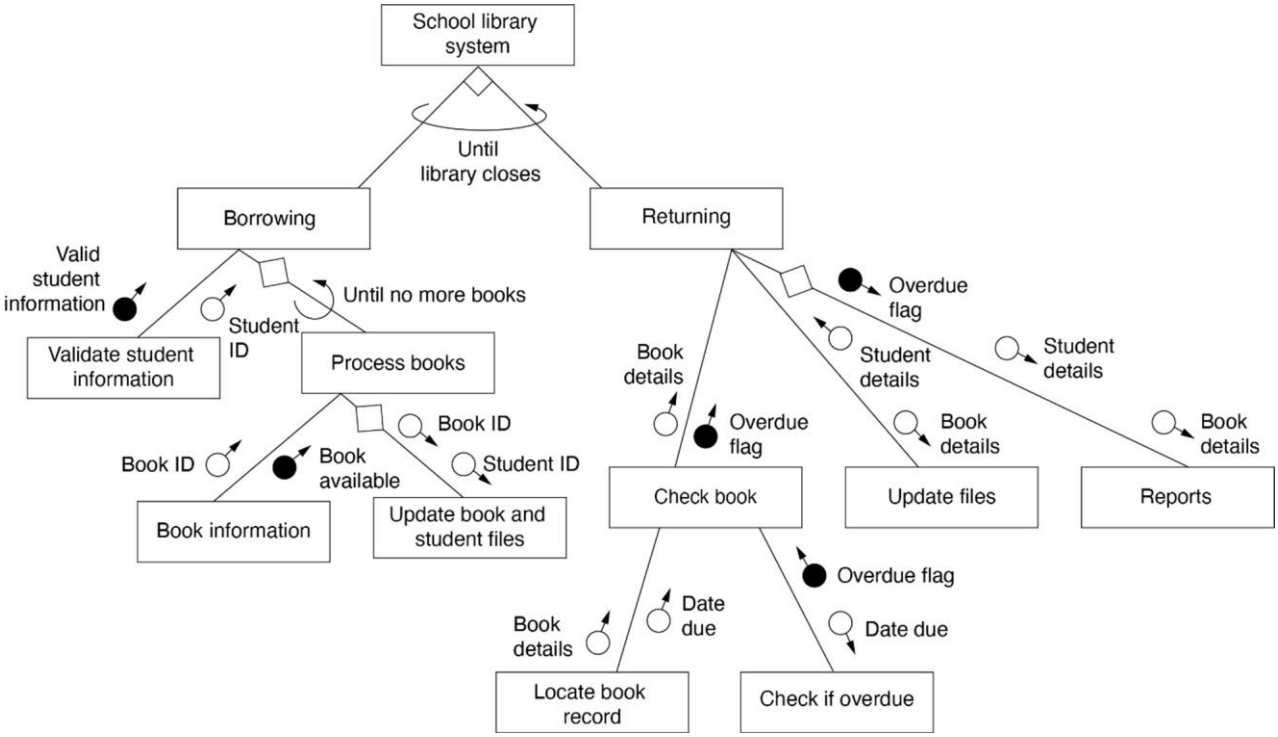
An empty circle is used to indicate data movement between subroutines (usually passed as parameters).

A filled circle is used to indicate a flag or control variable that is passed between subroutines.

A decision (ie optional execution of a subroutine) is indicated by use of a small diamond at the intersection of the connecting lines between subroutines that are called as the result of a binary or multi-way selection. Alternatively, the diamond may appear on a single connecting line if calling that subroutine is optional. In the diagram shown, a report may not necessarily be produced each time a book is returned.

Repetition (execution of a particular subroutine or set of subroutines multiple times) is shown by a curved arrow.

The following structure chart represents a library system.



Further detail for each of the lower-level subroutines can be shown in a separate structure chart, using the same name as the subroutine used in the main structure chart.

This method of providing successively more detail as required is known as *refinement*.

Data dictionary

A data dictionary provides a comprehensive description of each variable stored or referred to in a system. This commonly includes variable name, data type, format, size in bytes, number of characters to display the item including number of decimal places (if applicable), the purpose of each variable and a relevant example. Any validation rules applicable to the data item can also be included.

Details of records or arrays of records can be included in data dictionaries.

An extract of a data dictionary is shown.

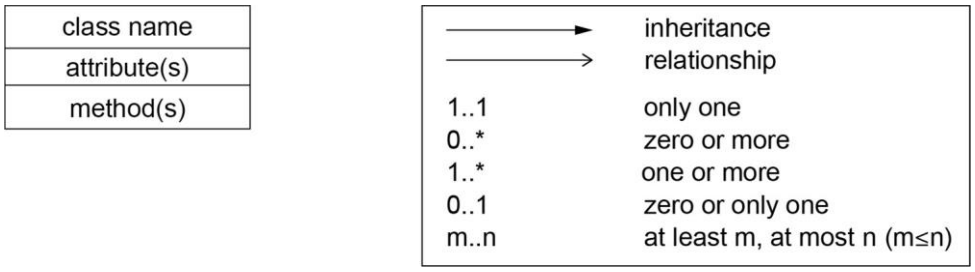
Variable	Data type	Format for display	Size in bytes	Size for display	Description	Example	Validation
UserId	String	XXNNN	5	5	A primary key, uniquely identifies user. First 2 letters of surname followed by unique 3-digit identifier	PT173	First 2 characters letters, followed by last 3 characters digits
UserName	String	XX..XX	15	15	Username of employee	Kim	First letter is a capital letter
DOB	Date and Time	YYYY/MM/DD	4	10	Birth date of employee	1953/10/05	Valid date less than today
Times_Late	Integer	NNN	2	3	Number of times late to work this year	147	Integer between 0 and 999
PayRate	Floating Point	$NNN.NN	4	7	Hourly rate of pay	$124.37	Decimal greater than 20, less than 400
SocialClub	Boolean	X	1 bit	1	Y or N	N	
Departments	Array (string)		20 * number of departments	N/A	Names of departments in organisation	Administration Finance Marketing	From a drop-down list

Note: a date and time data type is always stored as 32 bits (4 bytes) and can be displayed using different formats such as DD/MM/YYYY hh:mm:ss or YYYY/MM/DD

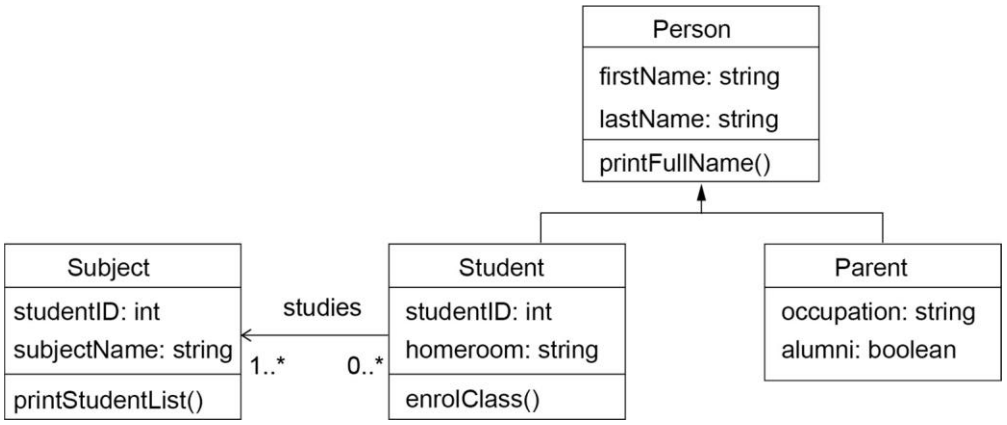
Class diagrams

Class diagrams provide a visual representation of systems that are implemented using the object-oriented paradigm. They model classes, their attributes and methods, and the relationships between classes.

Symbols



The following is an example of a class diagram.

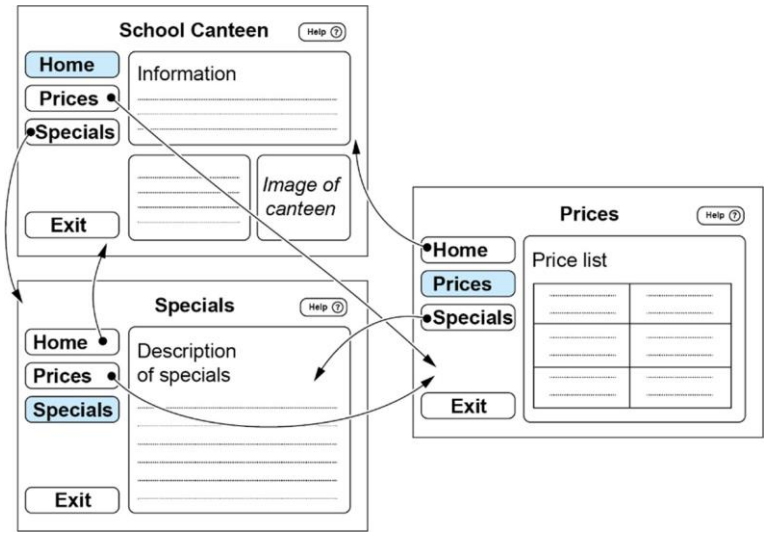


Note: Both Student and Parent inherit from Person. There is a relationship between Student and Subject. A student must study 1 or more subjects. A subject can have 0 or more students enrolled.

Storyboard

A storyboard shows the various interfaces (screens) as well as the links between them.

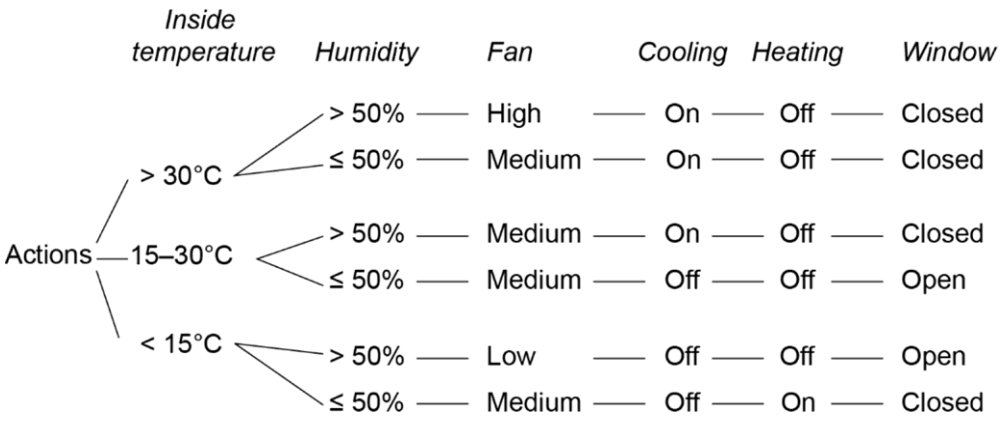
The following storyboard shows the relationship between three pages of information aimed at promoting a school canteen on a website.



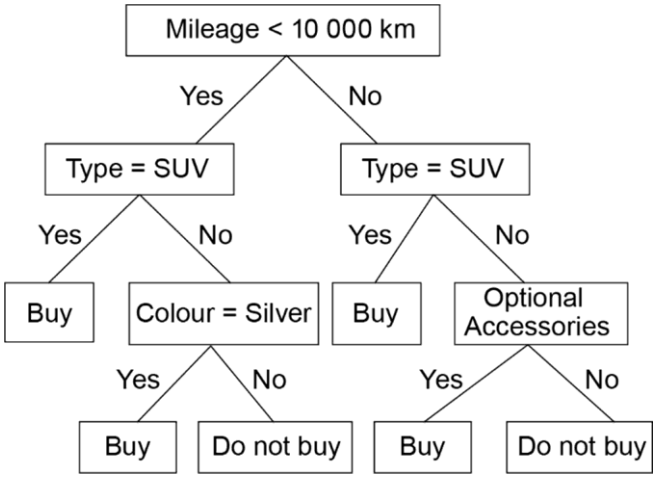
Decision trees

A decision tree is a diagram that represents all possible combinations of decisions and their resulting actions. Branches are shown to describe the eventual action depending on the condition at the time. Each decision path will lead to either another decision or a final action.

The following decision tree shows the rules in controlling the temperature system within a 'smart' house.



The following diagram shows another way to represent a decision tree.

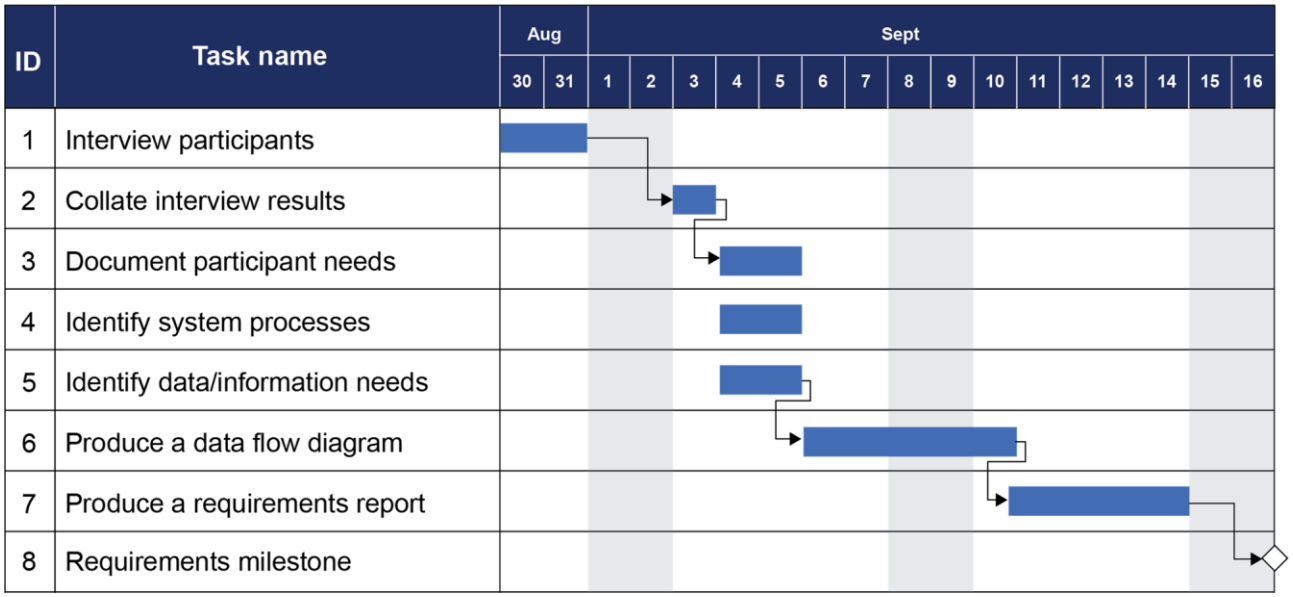


Project Management Tools

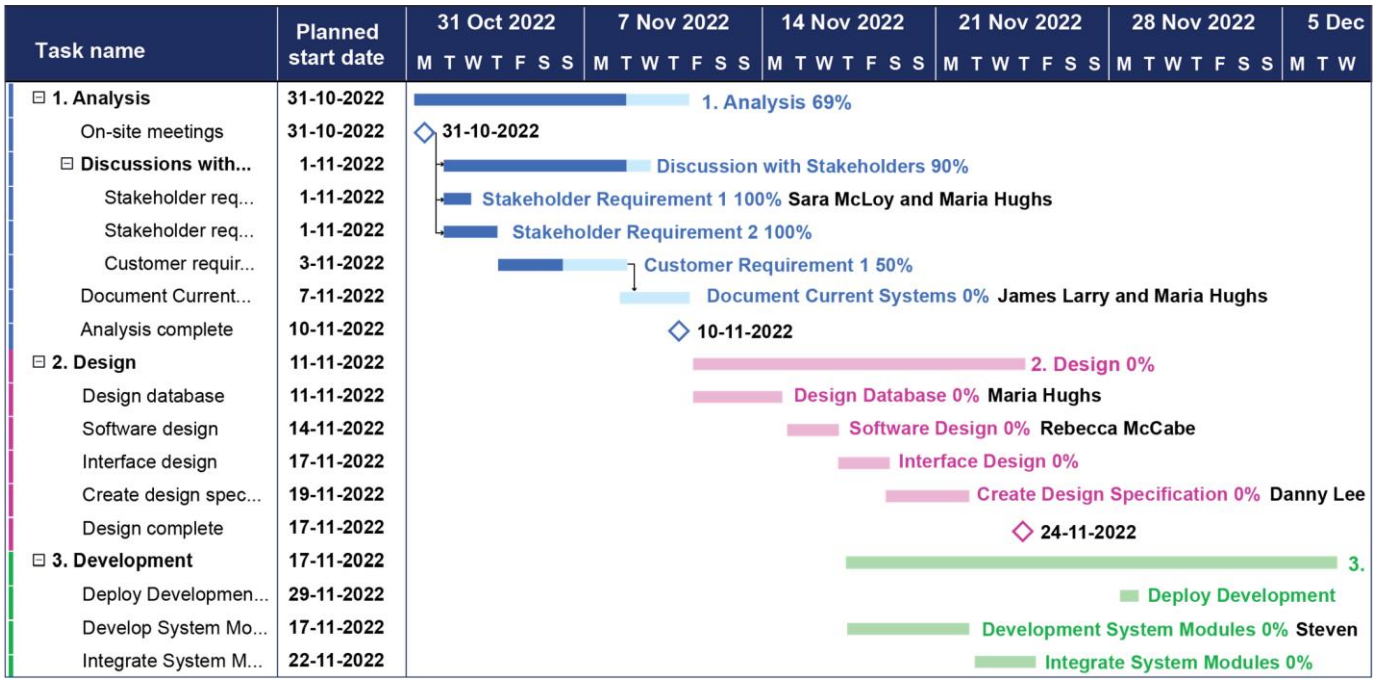
Gantt charts

A Gantt chart displays each of the component tasks in a proposed system development on an estimated timeline. Tasks should be named with self-explanatory titles. The estimated time required for each task and its dependent tasks should be clearly shown. The time scale should be clearly indicated with dates and important milestones in the project clearly marked.

The following diagram shows the main elements of a Gantt chart. Other formats are acceptable.



Gantt charts can also be used to allocate resources, including team members, to specific tasks. The following chart shows the percentage completion of tasks by each team member. Charts should be regularly updated during development to reflect actual versus estimated times for tasks.



Process diaries / log books

Process diaries / log books are used to document the progress of a project. Entries made by team members at regular intervals should include:

- date
- person making the entry
- progress since the last entry
- tasks achieved
- stumbling blocks or issues encountered and how they were managed
- possible approaches for upcoming tasks
- reflective comments
- resources used.

Programming Paradigms

Object-oriented paradigm

Students should know how to:

- define classes, objects, attributes and methods
- make use of inheritance, polymorphism and encapsulation
- perform message passing
- use control structures and variables.

Logic paradigm

Students should know how to:

- define and edit facts
- create, edit and remove rules
- display the solution and the rules that the system used.

Imperative paradigm

Students should know how to:

- use control structures and variables
- use assignment statements
- use expressions
- use subroutines.

Functional paradigm

Students should know how to:

- call functions and use recursion
- use functions as first-class objects and collections
- use abstraction, encapsulation, inheritance and polymorphism.

Note: Students should know how to use appropriate data structures for each of the paradigms.

Algorithms

It is expected that students are able to develop and interpret algorithms represented as pseudocode and flowcharts.

It is important to start complex algorithms with a clear, uncluttered mainline. The mainline should reference required subroutines, the details of which are shown in separate algorithms.

Each subroutine should be concise and correctly make use of further subroutines for detailed logic.

Pseudocode

Pseudocode is a method of describing the logic in an algorithm. It makes use of capitalised keywords and indentation to show control structures used.

In pseudocode:

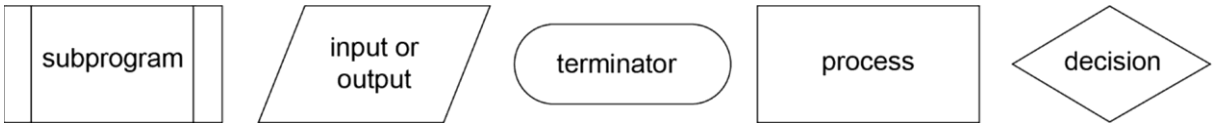
- keywords are written in capitals
- structural elements come in pairs, eg for every BEGIN there is an END, for every IF there is an ENDIF
- indenting is used to identify control structures in the algorithm
- when refining the solution to a problem, a subroutine can be referred to in an algorithm by its name, with a separate subroutine developed with that same name to show the detailed logic.

Flowcharts

Flowcharts are diagrams that represent algorithms and are read from top to bottom and left to right.

Symbols

Flowcharts use the following symbols connected by lines with arrowheads to indicate the flow of data. It is common practice to show arrowheads to avoid ambiguity.



Flowcharts using these symbols should be developed using only the standard control structures (as described in the following section, Control Structures).

Control Structures

Algorithms are developed using the basic control structures of *sequence*, *selection* and *repetition*. Students are expected to design algorithms and write code incorporating combinations of these control structures.

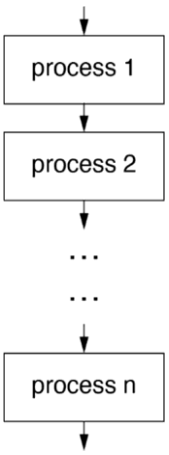
Sequence

Sequence refers to steps which are to be executed one after the other. The steps are executed in the same order in which they are written.

Pseudocode

```
process 1

process 2

…

…

process n
```

Flowchart



Selection

Binary selection

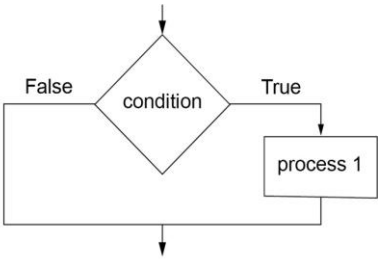
In binary selection, if the condition is met then one path is taken, otherwise the second possible path is followed.

Pseudocode

```
1.  IF condition THEN
        process 1
    ENDIF
```

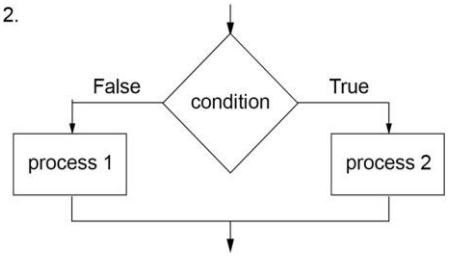
Flowchart

1.




```
2.  IF condition THEN
        process 2
    ELSE
        process 1
    ENDIF
```

2.



Note: Arrows coming from a decision symbol should be labelled to remove ambiguity.

Multi-way selection

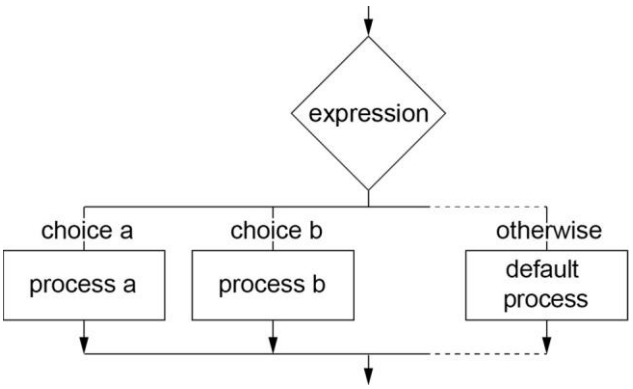
In multi-way selection there can be a number of possible choices, or cases. The path taken is determined by the evaluation of the expression. Once a relevant path has been determined and executed, execution of this expression ceases. Only one process is executed as a result of the implementation of the multi-way selection.

Multi-way selection is often referred to as a *case structure*.

Pseudocode

```
CASEWHERE expression evaluates to
    choice a: process a
    choice b: process b
    …
    OTHERWISE: default process
END CASE
```

Flowchart



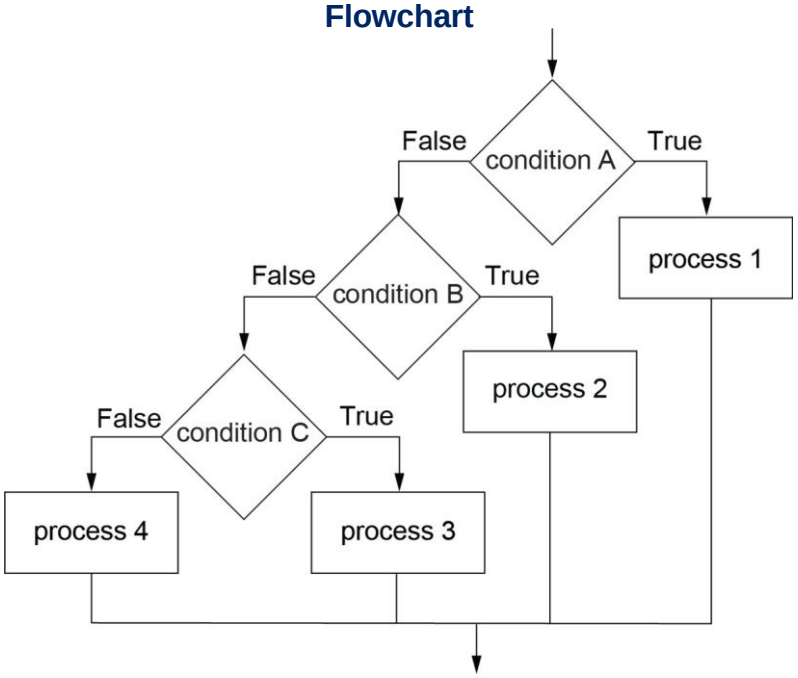
Nested IF

A nested IF allows the testing of multiple conditions with only one process executed.

Pseudocode

```
IF condition A THEN
       process 1
ELSEIF condition B THEN
       process 2
ELSEIF condition C THEN
       process 3
ELSE
       process 4
ENDIF
```

Flowchart



Repetition

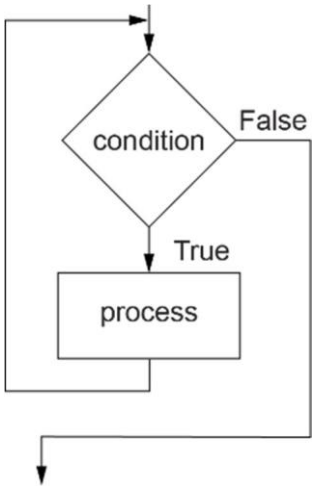
Pre-test

The pre-test loop tests the condition at the start of the loop to determine whether the body of the loop is executed. The body of the loop is executed repeatedly while the termination condition is true.

Pseudocode

```
WHILE condition is true
        process
ENDWHILE
```

Flowchart



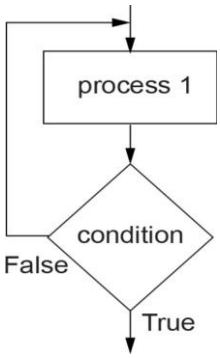
Post-test

A post-test loop executes the body of the loop before testing the termination condition. The body of the loop is repeatedly executed until the termination condition is true.

Pseudocode

```
REPEAT
        process

UNTIL condition is true
```

Flowchart



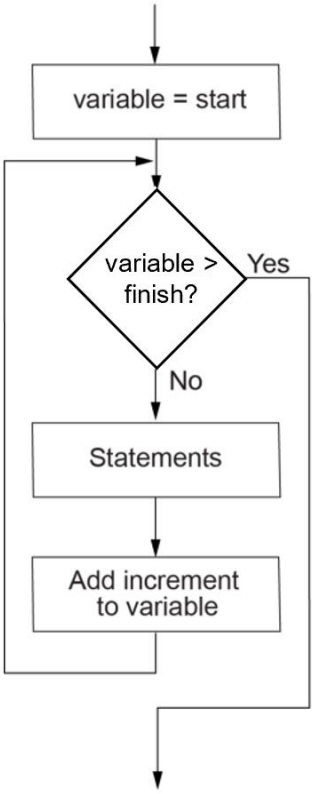
FOR / NEXT

FOR / NEXT loops (also known as counted loops) can be regarded as special cases of repetition and, depending on the language, are implemented as either pre-test or post-test repetitions.

Pseudocode

```
FOR variable = start TO finish STEP increment

       statements

NEXT variable
```

Flowchart



Note: Increment can take either a positive or negative value. The flowchart shows the logic for a positive increment

Subroutines

The terms *subroutine*, *module*, *subprogram* and *procedure* can be used interchangeably to represent a collection of statements that achieve a specific purpose.

A subroutine can do the same task at different points in an algorithm. It may operate on different data each time it is called. One or more parameters are used to indicate the data to be processed.

A *function* is a particular type of subroutine that returns a single value.

Using a subroutine with one parameter

The following algorithms represent the logic required to fill an array with characters.

The subroutine *read* uses a single parameter *arrayname* that can take different values.

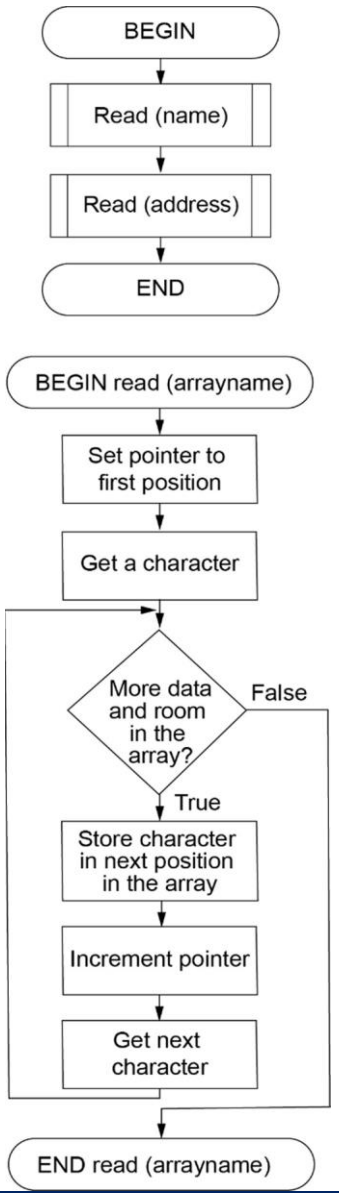
The first time that this subroutine is called, the data is read and stored in the array called 'name'. The second time, the data is read and stored in the array called 'address'.

Pseudocode

```
BEGIN

       read (name)

       read (address)

END
```

```
BEGIN read (arrayname)

       Set pointer to first position

       Get a character

       WHILE more data AND space in array

              Store data in arrayname at the
              position given by the pointer

              Increment the pointer

              Get next character

       ENDWHILE

END read (arrayname)
```

Flowcharts



Using a subroutine with multiple parameters

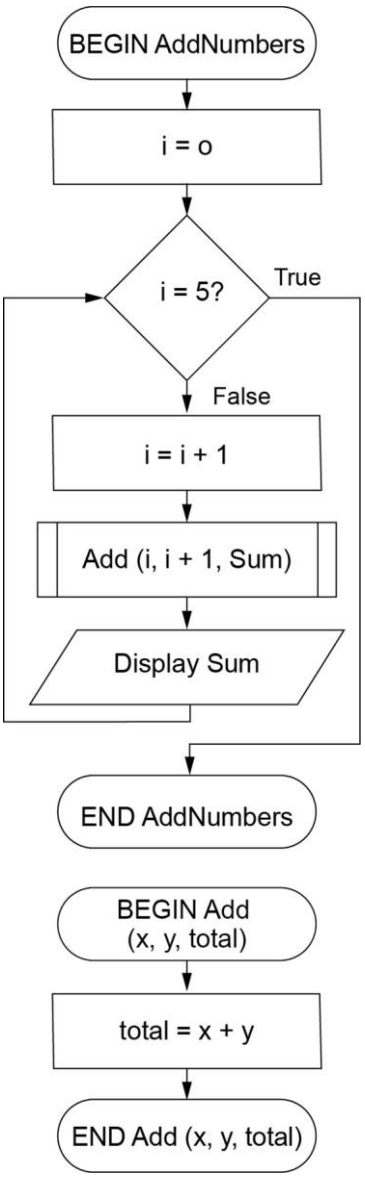
The following algorithms represent the logic required to display the sum of two consecutive integers, where the smaller integer takes all possible values from 1 to 5. The subroutine *Add* uses three parameters, *x, y* and *total.*

Pseudocode

```
BEGIN AddNumbers

    FOR i = 1 to 5

        Add (i, i + 1, sum)

        Display sum

    NEXT i

END AddNumbers
```

```
BEGIN Add (x, y, total)

    total = x + y

END Add (x, y, total)
```

Flowcharts



Passing a value back from a function

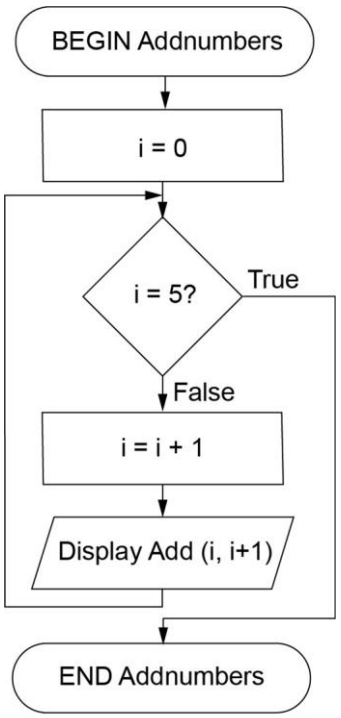
A function generates a single value. The word RETURN is used to pass this single value back from the function.

The algorithm *Addnumbers* uses the function *Add* to calculate the sum of two consecutive integers where the smaller integer takes all possible values from 1 to 5. In this case the function requires two parameters.

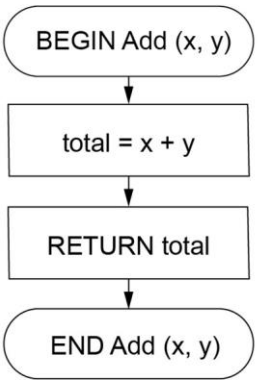
Pseudocode

```
BEGIN Addnumbers

    FOR i = 1 to 5

        Display Add (i, i + 1)

    NEXT i

END Addnumbers
```

Flowcharts




```
BEGIN Add (x, y)

    total = x + y

    RETURN total

END Add (x, y)
```



Relational Databases

SQL

Structured Query Language (SQL) is a language used to access and manipulate data in relational databases.

For the HSC Software Engineering course, the following syntax is used.

```
SELECT  the field(s) or calculated values to be displayed
FROM    the table(s) to be used
WHERE   the search criteria
GROUP BY    the field(s) used to group the returned rows
ORDER BY    the field(s) that determine the sequence of the displayed results
```

The keyword `AS` may be used within a `SELECT` statement to rename fields for display.

The search criteria may use relational operators, including the following.

```
CONTAINS
DOES NOT CONTAIN
EQUALS
NOT EQUAL TO
GREATER THAN
GREATER THAN OR EQUAL TO
LESS THAN
LESS THAN OR EQUAL TO
```

Logical operators that may be used include the following.

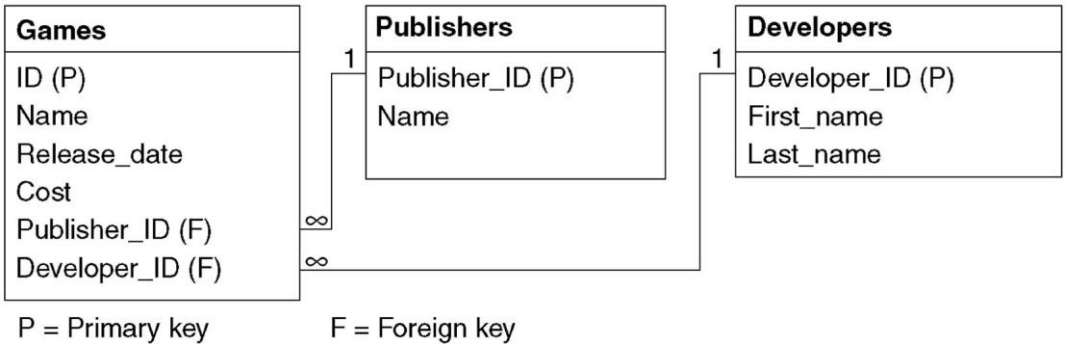
```
AND
OR
NOT
```

If the `GROUP BY` clause is used, it is useful to include in the `SELECT` statement details of the value to be calculated. The following functions may be used.

```
SUM (attribute)
AVG (attribute)
COUNT (attribute)
MAX (attribute)
MIN (attribute)
```

If the `ORDER BY` clause is used, the field and method should be specified. The methods used are typically identified by `ASC` (ascending: A – Z or 0 – 9) or `DESC` (descending: Z – A or 9 – 0).

Three tables from a relational database are shown.



The following query displays the name and release date of all games released from 1 March 2022 to 31 March 2023. The results will be displayed in ascending alphabetical order by game name.

```
SELECT Name, Release_date
FROM Games
WHERE Release_date >= '01/03/2022' AND Release_date <= '31/03/2023'
ORDER BY Name ASC
```

The following query displays each developer, together with the total cost of games they have developed for the publisher 'Games Inc', listed in descending order of developer name.

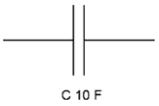
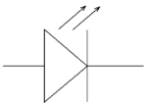
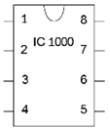
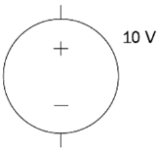
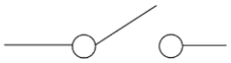
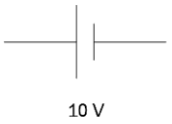
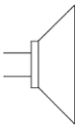
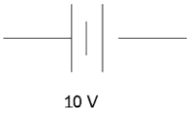
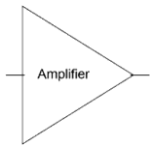
```
SELECT Developers.First_name, Developers.Last_name, SUM (Games.cost) AS Totalcost
FROM Games, Publishers, Developers
WHERE Publishers.Name = 'Games Inc'
AND Publishers.Publisher_ID = Games.Publisher_ID
AND Developers.Developer_ID = Games.Developer_ID
GROUP BY Developers.Developer_ID
ORDER BY Developers.Last_name DESC
```

Object-Relational Mapping (ORM)

ORM (Object-Relational Mapping) provides a layer of abstraction between the database and the programming language which the developer is using. In an object-oriented language, an ORM will usually be able to represent database items (rows) as objects in that chosen language and the attributes of that item (columns) as properties of the object. Students are expected to interpret code fragments used in an ORM framework.

Wiring diagrams for mechatronic systems

Mechatronic systems can range in complexity and can be represented using the following symbols.

Name	Symbol	Name	Symbol
Capacitor		LED	
Diode		Lightbulb	
Resistor		Integrated circuit	
2-way Switch		Voltage source	
On/off Switch		DC Voltage Source	
Speaker		DC Voltage source	
Motor		Amplifier	

If a wiring diagram requires other electrical components, the components must be clearly labelled for identification.

Programming for the Web

Front-end web development frameworks

There are numerous front-end web development frameworks which provide different features and benefits. Students should understand why such frameworks are useful in front-end web development

Students are not expected to have knowledge of a specific framework nor are they expected to code using any specific framework.

Cross-site scripting

Cross-site scripting (XSS) involves injecting malicious code into an otherwise safe website. It is usually done through user input that is not sufficiently sanitised before being processed and stored on the server.

Students should be able to interpret fragments of JavaScript related to cross-site scripting.

Cascading Style Sheets (CSS)

Cascading style sheets (CSS) are used to describe the formatting of web pages. Students are expected to interpret code fragments written in CSS and HTML. The following syntax is used.

```
/* Comment */
selector {
    property: value;
}
```

- **Comments:** can be specified anywhere in CSS and are enclosed in `/*` and `*/`
- **Selector:** targets a particular HTML element on the page to which the styling within the braces should be applied
- **Property:** a styling property, such as `color`
- **Value:** the value of the styling property, such as `red`

The following HTML fragment shows the CSS to style a webpage.

```
<html>
   <head><title>My Website</title></head>
   <body>
      <h1>Welcome!</h1>
      <p id=”welcome”>Welcome to my website!</p>
      <p class=”red-text”>This text should be red</p>
      <p>My website also has <span class=”red-text”>red text</span> here</p>
   </body>
</html>
```

The CSS to style that page is as follows.

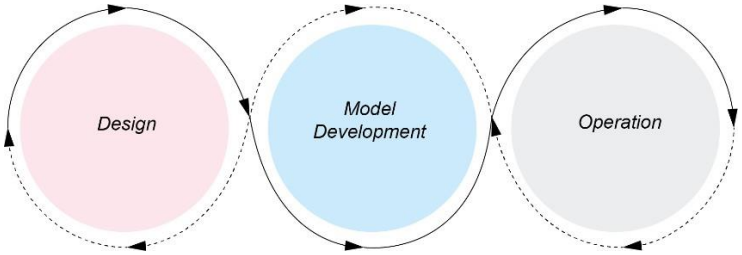
```
/* This selector targets an HTML element */
h1 {
   font-size: 18px;
}

/* This selector targets an HTML element with a specific “id” */
#welcome {
   font-style: italic;
}

/* This selector targets an HTML element with a specific “class” */
.red-text {
   color: red;
}
```

Machine Learning

Machine learning automation through DevOps

MLOps is the automated process of designing, training and deploying machine learning models. It borrows many of the same principles and practices used in DevOps, bringing together the teams involved in developing machine learning models and the operational teams involved in deploying and supporting the models in production.

Students should know the three stages of MLOps.



- Design:
  - defining the business problem to be solved
  - refactoring the business problem into a machine learning problem
  - defining success metrics
  - researching available data.
- Model development:
  - data wrangling
  - feature engineering
  - model training
  - model testing and validation.
- Operations:
  - model deployment
  - supporting operations/use
  - monitoring model performance.

Regression algorithms

Linear regression and polynomial regression algorithms are used to predict values in a continuous range, such as integers. These regression algorithms are used for machine learning.

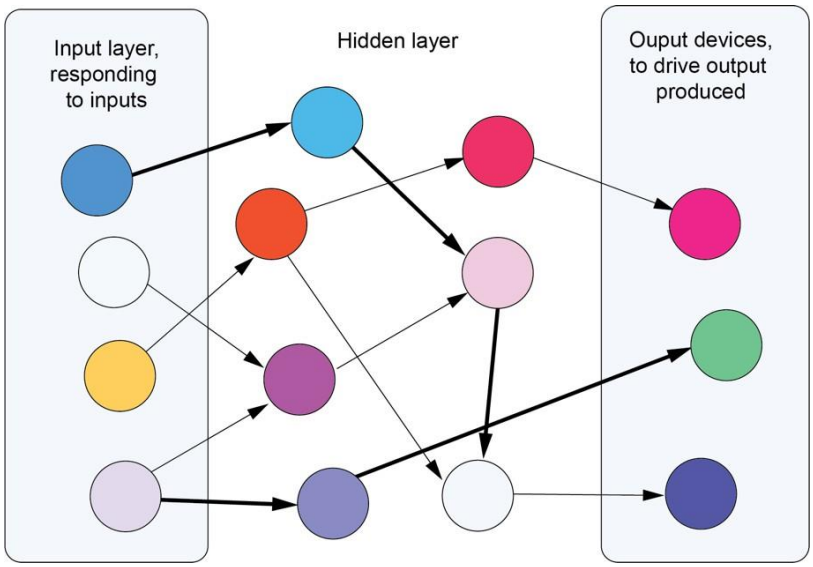
Logistic regression is used for classification problems.

Students should know how to design programs which use and apply these algorithms but are not expected to implement (or code) these complex algorithms.

The following Python code represents linear regression using NumPy and Scikit-learn machine learning frameworks.

```
# Import frameworks
import numpy
from sklearn.linear_model import LinearRegression

# Create the data for the two features
x = np.array([[2], [4], [6], [8], [10], [12], [14], [16]])
y = np.array([1, 3, 5, 7, 9, 11, 13, 15])

# Create the model
model = LinearRegression()

# Fit the model to the data (that is determine the line of best fit through the data)
model.fit(x, y)

# We can now use the model for predictions with existing or new data
# The model expects a value for x and will predict a value for y – so if we asked for a prediction on 4 it would return 3 (as that is known data).
y_prediction = model.predict(4)
>> 3

# If we asked for a prediction of 4.5 it should return 3.5 (even though that is unknown data we can tell what it should return, given there is a linear relationship)
y_prediction = model.predict(4.5)
>> 3.5
```

Neural Networks



Neural networks were designed to mimic the processing inside the human brain. They consist of a series of interconnected nodes (artificial neurones). Each neurone can accept a binary input signal and potentially output another signal to connected nodes.

Training cycle

Internal weightings and threshold values for each node are determined in the initial training cycle for each neural network. The system is exposed to a series of inputs with known responses. Linear regression with backward chaining is used to iteratively determine the set of unique values required for output. Regular exposure to the training cycle results in improved accuracy and pattern matching.

Execution cycle

In the diagram, signal strength between nodes with the strongest weightings are thicker representing a higher priority in determining the final output. The execution cycle follows the training cycle and utilises the internal values developed during the training cycle to determine the output.

Methods for Testing a System

Students are expected to know and understand the following methods for testing a software solution.

- functional testing
- acceptance testing
- live data
- simulated data
- beta testing
- volume testing.

Character Representation

Characters can be represented with ASCII or Unicode. When a developer is working with text strings, they can make use of how the text data is stored internally in its binary format to perform functions such as changing the case of letters in the string, or performing a simple encryption.

Programming with Python

Students are expected to be able to code using the Python programming language.

Students should be familiar with the use of the following features:

- control structures
- global and local variables
- use of simple and structured data types
- classes, objects, attributes and methods
- functions
- modules and libraries
- file handling

Students are expected to design and implement programs incorporating combinations of these features.