**Additional Syllabus Support Booklet**

Computer Science

ATAR Years 11 and 12

**Acknowledgement of Country**

Kaya. The School Curriculum and Standards Authority (the Authority) acknowledges that our offices are on Whadjuk Noongar boodjar and that we deliver our services on the country of many traditional custodians and language groups throughout Western Australia. The Authority acknowledges the traditional custodians throughout Western Australia and their continuing connection to land, waters and community. We offer our respect to Elders past and present.

**Copyright**

© School Curriculum and Standards Authority, 2022

This document – apart from any third-party copyright material contained in it – may be freely copied, or communicated on an intranet, for non-commercial purposes in educational institutions, provided that the School Curriculum and Standards Authority (the Authority) is acknowledged as the copyright owner, and that the Authority’s moral rights are not infringed.

Copying or communication for any other purpose can be done only within the terms of the *Copyright Act 1968* or with prior written permission of the Authority. Copying or communication of any third-party copyright material can be done only within the terms of the *Copyright Act 1968* or with permission of the copyright owners.

Any content in this document that has been derived from the Australian Curriculum may be used under the terms of the [Creative Commons Attribution 4.0 International licence](https://creativecommons.org/licenses/by/4.0/).

**Disclaimer**

Any resources such as texts, websites and so on that may be referred to in this document are provided as examples of resources that teachers can use to support their learning programs. Their inclusion does not imply that they are mandatory or that they are the only resources relevant to the course.

**Contents**

[Purpose 1](#_Toc137555059)

[Programming 1](#_Toc137555060)

[Software licensing 1](#_Toc137555061)

[Conventions for writing pseudocode 1](#_Toc137555062)

[Programming control structures 2](#_Toc137555063)

[Modularisation 4](#_Toc137555064)

[Object oriented programming 6](#_Toc137555065)

[Common algorithms 8](#_Toc137555066)

[Network communications 13](#_Toc137555067)

[Key protocols associated with layers in models 13](#_Toc137555068)

[Network diagram conventions (CISCO) 14](#_Toc137555069)

[Cyber security 16](#_Toc137555070)

[Types of malware 16](#_Toc137555071)

[Common methods of encryption 16](#_Toc137555072)

[Data management 17](#_Toc137555073)

[Entity relationship diagrams 17](#_Toc137555074)

[Data dictionaries 18](#_Toc137555075)

[Normalisation 18](#_Toc137555076)

[Common SQL 24](#_Toc137555077)

[Appendices 26](#_Toc137555078)

[Control Structure Python Examples 26](#_Toc137555079)

[Object Oriented Python Examples 28](#_Toc137555080)

# Purpose

This document is intended to support the delivery of the Year 11 and Year 12 Computer Science Australian Tertiary Admission Rank (ATAR) syllabuses. It contains conventions, standards, specifications and examples to provide teachers and students with clarity relating to the expected depth of teaching of some relevant content points in each syllabus.

# Programming

Python is the prescribed programming language for the Computer Science ATAR course and will be used in ATAR examination questions related to programming.

## Software licensing

* Proprietary
* Open source
  + Public domain
  + Permissive
  + GNU Lesser General Public License
  + Copy left

## Conventions for writing pseudocode

Although there is no specific format for writing pseudocode, the following conventions should be used in this course.

* Use capital letters for keywords.
* Indent lines of code to show the structure of the code and identify control structures; for example, commands in a loop should be indented.
* The end of structural elements and control structures should be explicitly indicated; for example, IF…END IF.
* Use the symbol = (a single equal sign) to indicate an assignment statement.
* Use the symbol == (two equal signs) to indicate a comparison statement.
* Initialise all variables at the start of each module.
* Clearly indicate constants using the CONST keyword.
* Clearly indicate global variables using the GLOBAL keyword.

### Common commands for writing pseudocode

| **Command** | **Pseudocode** |
| --- | --- |
| User input | INPUT(num) |
| User output | PRINT(“Hello world!”) |
| Assignment | = |
| Equals (comparison) | == |
| Not equal to | != |
| Greater than | > |
| Greater than or equal to | >= |
| Less than | < |
| Less than or equal to | <= |
| Integer division | DIV or // e.g. 7 // 2 = 3 |
| Modulus (remainder) | MOD or % e.g. 7 % 2 = 1 |
| OR | x < 1 OR x > 10 |
| AND | x > 1 AND x < 10 |
| Arrays | scores = []  scores[0] = 15  scores[1] = 16  scores.append(12) # add element to end of array  scores.length # gives the number of elements in an array |
| Dictionaries | costOfGear = {  “mask”: 2,  “wetsuit: 5,  “BCD”: 5,  “tank”: 5  }  costOfGear[“fins”] = 2 # add new key:value pair  costOfGear[“wetsuit”] = 6 # update value of wetsuit  cost = costOfGear[“mask”] # value of cost will be 2  costOfGear.keys # list of all keys in the dictionary  costOfGear.values # list of all values in the dictionary  costOfGear.items # list of all key/value pairs in the dictionary |

## Programming control structures

| **Structure** | **Example** |
| --- | --- |
| **Sequence**: | INPUT(num1)  INPUT(num2)  product = num1 \* num2  PRINT(product) |
| **One-way selection:** |  |
| IF condition THEN  do something  END IF | speed = 50  IF speed > 50 THEN  PRINT(“You are speeding”)  END IF |
| **Two-way selection:** |  |
| IF condition THEN  do something  ELSE  do something  END IF | speed = 50  IF speed > 50 THEN  PRINT(“You are speeding”)  ELSE  PRINT(‘You are not speeding”)  END IF |
| **Multi-way selection:** |  |
| **Method 1 – IF…ELSE IF…ELSE**  IF condition THEN  do something  ELSE IF condition THEN  do something  ELSE  do something  END IF  **Method 2 – CASE statement**  CASE value OF  choice 1: do something  choice 2: do something  OTHERWISE: do something  END CASE | **Method 1 – IF…ELSE IF…ELSE**  speed = 50  IF speed < 20 THEN  PRINT(“You are going too slow”)  ELSE IF speed > 50 THEN  PRINT(‘You are speeding”)  ELSE  PRINT(‘You are not speeding”)  END IF  **Method 2 – CASE statement**  colour = ‘red’  CASE colour OF  ’red’: PRINT(“Stop”)  ‘yellow’: PRINT (“Slow down”)  ‘green’: PRINT(“Go”)  OTHER: PRINT(“Incorrect colour”)  END CASE |
| **Test-first loop (WHILE)** |  |
| WHILE condition is True  do something  END WHILE | num = 0  WHILE num < 10  PRINT(“The number is “ + num)  num = num + 1  END WHILE |
| **Test-last loop (REPEAT UNTIL)** |  |
| REPEAT  do something  UNTIL condition I True | REPEAT  INPUT(Age)  UNTIL (Age >= 6) AND (Age <= 17)  PRINT (Age) |
| **Fixed loop (FOR)** |  |
| FOR variable = start TO finish [STEP increment]  do something  END FOR | FOR num = 1 TO 10  PRINT(“The number is “ + num)  END FOR  FOR num = 10 TO 1 STEP –1  PRINT(num)  END FOR  PRINT("Blast off!”)  FOR num = 1 TO 100 STEP 10  PRINT(“The number is “ + num)  END FOR |

## Modularisation

Modularisation is a methodology that involves breaking a problem down into smaller, less complex parts. Benefits of modularisation include:

* it allows code to be reused and reduced code repetition
* it allows more people to work on a project – each person can work on separate modules
* it breaks a large complex problem down into smaller problems to make it easier to solve
* it makes it easier to read algorithms and programs
* it makes it quicker and easier to find errors.

As in most modern programming languages, there is no distinction made between modules and functions in the ATAR syllabus – the two terms can be used interchangeably in pseudocode. When a value needs to be returned from a module, then the RETURN keyword should be used.

Good programming practice suggests that a function should perform a single task, and where necessary return a single value using the RETURN keyword. The use of reference parameters in place of returning a value from a function should be avoided wherever possible.

| **Without Modularisation** | **With Modularisation** |
| --- | --- |
| FUNCTION Main  INPUT(length)  INPUT(height)  area1 = length \* height  INPUT(length)  INPUT(height)  area2 = length \* height  INPUT(length)  INPUT(height)  area3 = length \* height  total = area1 + area2 + area3  PRINT(“The total area is”, total)  END Main | FUNCTION Main  INPUT(length)  INPUT(height)  area1 = CalculateArea(length, height)  INPUT(length)  INPUT(height)  area2 = CalculateArea(length, height)  INPUT(length)  INPUT(height)  area3 = CalculateArea(length, height)  total = area1 + area2 + area3  PRINT(“The total area is”, total)  END Main  FUNCTION CalculateArea(length, height)  area = length \* height  RETURN area  END CalculateArea |

The code on the left repeats the same lines of code three times where it calculates the area based on the length and height. The code on the right reduces this repetition by moving those lines of code to a separate module.

### Parameters

We use parameters to pass values between functions. There are two types of parameters.

* **Value parameters**: a copy of the actual data is passed to the function that is being called. Any changes to the parameter inside the function do not affect the original value.
* **Reference parameters**: a pointer to the variable’s memory location is passed to the function being called. Any changes to the parameter cause the original value to be changed.

In most programming languages, simple data types will be passed by value, and complex data types (such as arrays and records) will be passed by reference.

To indicate a parameter is a reference parameter, it is suggested that the REF keyword is used. A reference parameter would be used when passing a variable that could be quite large (such as a list of objects). For example:

FUNCTION DoSomethingToMyList(REF bigList)

END DoSomethingToMyList

aReallyBigList = [obj1, obj2, … , obj1000]

DoSomethingToMyList(aReallyBigList)

# Object-oriented programming

Object-oriented programming (OOP) programs are based around the data that is needed and the operations that need to be performed on that data, rather than the procedural logic of the program.

**Classes**: user-defined template that represents an object. This defines the attributes of each object and the methods that can be performed.

**Objects**: specific instances of a class using data for that instance.

**Attributes**: data stored about each object that show the current state of the object.

**Methods**: functions defined in the class that define the behaviours of the object.

### Creating a new class

CLASS Animal

Attributes:

name

hunger = 5

food\_list = []

Methods:

FUNCTION Animal(new\_name)

name = new\_name

END Animal

FUNCTION eat(food)

result = ""

IF food IN food\_list

result = "Not hungry"

IF hunger > 0

hunger = hunger - 1

result = "That was yummy"

END IF

ELSE

result = "I don't like that food"

END IF

RETURN result

END eat

FUNCTION is\_hungry()

RETURN hunger > 0

END is\_hungry

END Animal

### Instantiating and using an object:

Instantiation refers to creating a specific object from a class that can be used in your program.

horse = new Animal(“Silver”) #Creates a horse with the name "Silver"horse.food\_list.append(“grass”) # Will add grass to the food\_list

horse.eat(“potato”) # Will return “I don’t like that food”

### Inheritance

One of the powerful features of OOP is that it allows the programmer to easily re-use code by classifying objects and inheriting common features from a base class. For example, a dog is a type of animal that has the base attributes of hunger and food\_list. The Dog class sets a default food\_list specific to dogs and adds two new attributes, has\_fur and legs.

CLASS Dog : Animal

Attributes:

has\_fur = True

legs = 4

food\_list = [“meat”, “bones”]

Methods:

FUNCTION bark()

RETURN name + “is barking”

END

FUNCTION number\_of\_legs()

RETURN legs

END number\_of\_legs

END Dog

CLASS Fish : Animal

Attributes:

has\_fins = True

food\_list = [‘algae’, ‘plankton’]

Methods:

FUNCTION swim()

RETURN name + ‘is swimming’

END swim

END Fish

Fido = new Dog()

PRINT(fido.number\_of\_legs())

Goldie = new Fish()

PRINT(goldie.has\_fins)

## Common algorithms

### Arrays

#### Load an array

FUNCTION LoadArray

name = “”

i = 0

names = []

PRINT(“Enter a name: ”)

INPUT(name)

WHILE name != “”

names[i] = name

i = i + 1

INPUT(name)

END WHILE

PRINT(“There were”, i, “names entered.”)

END LoadArray

#### Print contents of an array

FUNCTION PrintArray

names = [“Peter”, “Jane”, “Hugo”, “Kai”, “Sally”, “Arman”]

FOR i = 0 TO names.length – 1

PRINT names[i]

END FOR

END PrintArray

#### Add contents of an array

FUNCTION AddArray

numbers = [4, 8, 23, 52, 3, 27, 86]

total = 0

FOR i = 0 TO numbers.length – 1

total = total + numbers[i]

END FOR

PRINT(total)

END AddArray

#### Minimum value in array

FUNCTION FindMinimumValue

numbers = [4, 8, 23, 52, 3, 27, 86]

min = numbers[0]

minIndex = 0

FOR i = 1 TO numbers.length – 1

IF numbers[i] < min THEN

min = numbers[i]

minIndex = i

END IF

END FOR

PRINT(“The minimum value is”, min)

PRINT(“The minimum value is at position”, minIndex)

END AddArray

#### Maximum value in array

FUNCTION FindMaximumValue

numbers = [4, 8, 23, 52, 3, 27, 86]

max = numbers[0]

maxIndex = 0

FOR i = 1 TO numbers.length – 1

IF numbers[i] > max THEN

max = numbers[i]

maxIndex = i

END IF

END FOR

PRINT(“The maximum value is”, max)

PRINT(“The maximum value is at position”, maxIndex)

END AddArray

### File processing

FUNCTION ReadFile

myfile = OPEN\_READ(“data.txt”)

lines = []

WHILE NOT myfile.EOF

line = myfile.READLINE()

lines.append(line)

END WHILE

CLOSE(myfile)

END ReadFile

FUNCTION WriteFile

myfile = OPEN\_WRITE(“ouputfile.txt”)

lines = [“Twinkle Twinkle Little Star”, “Baa Baa Black Sheep”, “Hickory Dickory Dock”]

FOR i = 0 TO (lines.length – 1)

myfile.WRITELINE(lines[i])

END FOR

CLOSE(myfile)

END WriteFile

FUNCTION AppendFile

myfile = OPEN\_APPEND(“names\_file.txt”)

names = [“James Smith”, “Aaron Jones”, “Sally Gonzales”]

FOR i = 0 TO (names.length – 1)

myfile.WRITELINE(names[i])

END FOR

CLOSE(myfile)

END WriteFile

### Search algorithms

#### Linear search

The linear search will go through an array and check each element for the target until it is found. If it does not find the target, it will move through the array until the end.

The algorithm below will return the index of the target element if it is found. If the target element is not found it will return -1.

FUNCTION LinearSearch(searchArray, target)

index = 0

position = -1

WHILE index < searchArray.length AND position == -1

IF searchArray[index] = target THEN

position = index

END IF

index = index + 1

END WHILE

RETURN position

END LinearSearch

#### Binary search

The binary search works by comparing the middle element of an array to the target element. If a match is not found, then the element array is split into two. If the element is less than the middle element, then the sub-array continues the search until the numbers can be split.

Note: The binary search requires the array to be sorted to work properly.

FUNCTION BinarySearch(searchArray, target)

position = -1

lowerBound = 0

upperBound = searchArray.length – 1

WHILE lowerbound <= upperBound AND position == -1

midpoint = (lowerBound + upperBound) / 2

IF searchArray[midpoint] < target THEN

lowerBound = midpoint + 1

ELSE IF searchArray[midpoint] > target THEN

upperBound = midpoint – 1

ELSE

position = midpoint

END IF

END WHILE

RETURN position

END BinarySearch

### Sort algorithms

#### Bubble sort

FUNCTION BubbleSort(arrayToSort)

last = arrayToSort.length - 1

swapped = TRUE

WHILE swapped

swapped = FALSE

i = 0

WHILE i < last

IF arrayToSort [i] > arrayToSort [i + 1] THEN

temp = arrayToSort [i]

arrayToSort [i] = arrayToSort [i + 1]

arrayToSort [i + 1] = temp

swapped = TRUE

END IF

i = i + 1

END WHILE

last = last - 1

END WHILE

RETURN arrayToSort

END BubbleSort

#### Insertion sort

FUNCTION InsertionSort(arrayToSort)

position = 0

WHILE position < arrayToSort.length

currentValue = arrayToSort[position]

sortedPosition = position - 1

WHILE sorted\_position >= 0 and arrayToSort[sortedPosition] > currentValue

arrayToSort[sortedPosition + 1] = arrayToSort[sortedPosition]

sortedPosition = sortedPosition - 1

END WHILE

arrayToSort[sortedPosition + 1] = currentValue

position = position + 1

END WHILE

return arrayToSort

END InsertionSort

**Selection sort**

FUNCTION SelectionSort(arrayToSort)

unsortedIndex = arrayToSort.length – 1

WHILE unsortedIndex > 0

i = 0

max = arrayToSort[i]

maxIndex = i

WHILE i <= unsortedIndex

i = i + 1

IF arrayToSort[i] > max THEN

max = arrayToSort[i]

maxIndex = i

END IF

END WHILE

temp = arrayToSort[maxIndex]

arrayToSort[maxIndex] = arrayToSort[unsortedIndex]

arrayToSort[unsortedIndex] = temp

unsortedIndex = unsortedIndex - 1

END WHILE

RETURN arrayToSort

END SelectionSort

# 

# Network communications

## Key protocols associated with layers in models

The following table shows some of the key protocols associated with the different layers of the Department of Defence Transfer Communication Protocol/Internet Protocol (DoD TCP/IP) model.

|  |  |  |
| --- | --- | --- |
| **DoD TCP/IP model** | **OSI model** | **Key protocols** |
| Application | Application | SMTP, FTP, HTTP, HTTPS, DHCP, DNS, PING |
| Presentation |
| Session |
| Transport | Transport | TCP & UDP |
| Internet | Network | IPV6, IPv4, ARP |
| Network | Data Link | Ethernet (802.3), Wi-Fi (802.11) |
| Physical |

## Network diagram conventions (CISCO)

|  |  |  |
| --- | --- | --- |
| **Router** | Wired | Wireless |
| **Switch** |  | |
| **Wireless access point** |  | |
| **Firewalls** |  | |

### Diagram Description automatically generatedNetwork diagram example:

# Cyber security

## Types of malware

* Ransomware
* Viruses
* Rootkits
* Spyware
* Backdoors
* Phishing

## Common methods of encryption

### Early methods and weaknesses

* **Substitution cipher** swaps out characters. Assuming 26 alphabet characters, it is easily broken using character frequency.
* **Vigenère cipher** uses a repeated key combing plain text with the key. Easily broken if we know the length of the key and use the character frequency method similar to the substitution cipher.
* **Mechanical encryption** such as the World War II (WW2) Enigma machine. Each mechanical method had its own weakness. The Enigma’s weakness was it never encrypted a letter as itself.
* **Data Encryption Standard (DES)** was the first digital encryption standard used a key size of 56 bits. That is small compared to today’s standards and is quickly cracked with fast processing speeds available today.
* **Advanced Encryption Standard (AES)** replaced DES as the commonly used method of encryption. It uses 128, 192 and 256 bits and is yet to be cracked.

DES and AES use symmetric keys, which means the key used to encrypt is the same key to decrypt. This is a problem if you need to securely communicate with someone who does not have the private key. **RSA (Rivest–Shamir–Adleman)** encryption solves the problem with asymmetric encryption –data is encoded with a public key that is then decrypted using a private key. It is very slow compared to AES, so it’s often used to securely communicate the private AES key. RSA uses 2048–4096 key sizes and works using a key produced by an algorithm using two prime numbers.

### Current best practice

* Secure your private key – a stolen key means your data is no longer secure. Ensure only those who need the key are able to access it.
* Back up your key – a lost key means lost data as it will be permanently encrypted.
* Use longer length keys to ensure brute force cracking is harder.
* Use audit logs to check if keys have been accessed by unauthorised users.
* Best practice is that users should encrypt any messages, critical or sensitive files they send. This extends to the encryption of storage devices in case they fall into the wrong hands.
* Best practice is based upon the guidelines from NIST: (National Institute of Standards and Technology) <https://csrc.nist.gov/Projects/cryptographic-standards-and-guidelines>.

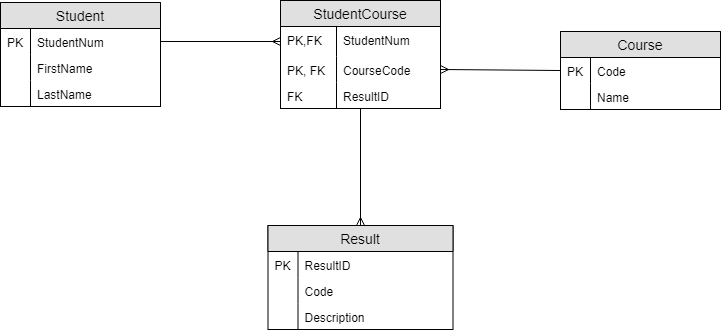
# Data management

## Entity relationship diagrams

An Entity relationship (ER) diagram provides a graphical representation of the relationships between the entities in a database. In this course, ER diagrams are to be drawn using crow’s foot notation as shown below.

|  |  |  |
| --- | --- | --- |
| **Entity and attributes** | **Relationships between entities** | |
|  |  | One to one  One to many  Many to many |

### ER diagram example



## Data dictionaries

Data dictionaries provide metadata that describes the attributes of data to be stored in a database.

Fields include:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Element Name | Data Type | Size | Description | Constraints |

### Data dictionary example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element Name** | **Data Type** | **Size** | **Description** | **Constraints** |
| StudentNum | Integer |  | Unique identifier for student | Must be unique and not null |
| GivenName | Text | 20 | Student’s given name | Not null |
| FamilyName | Text | 20 | Student’s family name | Not null |

NOTE: Description should include a brief description of the data being stored, the format of the data and the default value if applicable.

## Normalisation

Normalisation is the process of identifying and eliminating data anomalies and redundancies, thereby improving data integrity and efficiency for storage in a relational database. This process is designed to remove repeated data and improve database design.

### Data Anomalies

Consider the data in the table below. This unnormalised data can cause problems when data is updated, added or deleted.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Student Number** | **Given Name** | **Family Name** | **Email** | **Course** | **Course Name** |
| 10010504 | David | Rossi | [drossi@student.edu.au](mailto:dunaipon@student.edu.au) | MATH1001 | Mathematics 1A |
| 10010504 | David | Rossi | [drossi@student.edu.au](mailto:dunaipon@student.edu.au) | COMP1001 | Computing 1A |
| 10010504 | David | Rossi | [drossi@student.edu.au](mailto:dunaipon@student.edu.au) | MATH1002 | Mathematics 1B |
| 24352494 | Debbie | Tainton | [dtainton@student.edu.au](mailto:dtainton@student.edu.au) | MATH1002 | Mathematics 1B |
| 24352494 | Alison | Roach | [aroach@student.edu.au](mailto:aroach@student.edu.au) | MATH2001 | Mathematics 2B |

#### Update anomaly

An update anomaly occurs when you try to update data that is stored in multiple locations. If all records are not updated, then data could become inconsistent and/or inaccurate. For example, if David Rossi updates his email address, then all three occurrences need to be updated

#### Delete anomaly

A delete anomaly occurs when by deleting one piece of data you delete the only instance of another piece of data. For example, if Alison Roach was removed from the database, then we would also lose all the information about the subject Mathematics 2B.

#### Insert anomaly

An insert anomaly occurs when data cannot be added because only part of the data is available. For example, if a new subject is added, but no student allocated, then we would be unable to add the subject as we would not have all the necessary information to create a new record.

### Normalisation to 3NF (3rd Normal Form)

Steps to normalisation of data:

* 1. ensure data is in the form of a relation
  2. convert data to 1NF (1st Normal Form)
  3. convert data to 2NF (2nd Normal Form)
  4. convert data to 3NF (3rd Normal Form).

#### Converting data to a relation

For data to be in the form of a relation:

1. it must have no repeated attributes
2. all cells must be atomic (that is, they must only contain a single piece of data).

**Repeated Fields**

The following table is **not** in the form of a relation as it has repeating fields – the Course field is repeated multiple times.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Student Number** | **Given name** | **Family name** | **Course 1** | **Course 2** | **Course 3** |
| 10010504 | David | Rossi | MATH1001 | COMP1001 | MATH1002 |
| 24352494 | Debbie | Tainton | MATH1001 |  |  |

**Non-atomic Field**

The following table is **not** in the form of a relation as one of the fields is not atomic – the Course field for David Rossi has information about three different courses.

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Number** | **Given name** | **Family name** | **Course** |
| 10010504 | David | Rossi | MATH1001, COMP1001, MATH1002 |
| 24352494 | Debbie | Tainton | MATH1001 |

**Relation**

The following table **is** in the form of a relation as all fields are atomic and there are no repeating fields. This data is not normalised and would not make a good database structure, but we can now start the process of normalisation.

### Relation example

| **Student Number** | **Given name** | **Family name** | **Email** | **Course** | **Course Name** |
| --- | --- | --- | --- | --- | --- |
| 10010504 | David | Rossi | [drossi@student.edu.au](mailto:dunaipon@student.edu.au) | MATH1001 | Mathematics 1A |
| 10010504 | David | Rossi | [drossi@student.edu.au](mailto:dunaipon@student.edu.au) | COMP1001 | Computing 1A |
| 10010504 | David | Rossi | [drossi@student.edu.au](mailto:dunaipon@student.edu.au) | MATH1002 | Mathematics 1B |
| 24352494 | Debbie | Tainton | [dtainton@student.edu.au](mailto:dtainton@student.edu.au) | MATH1002 | Mathematics 1B |
| 24352494 | Alison | Roach | [aroach@student.edu.au](mailto:aroach@student.edu.au) | MATH2001 | Mathematics 2B |

### Process of normalisation:

#### 1NF (1st Normal Form)

To be in 1st Normal Form, we must:

1. ensure that all fields are atomic
2. remove all repeating attributes.

Each relation that is formed will have a primary key. Primary keys are indicated with the use of underlining the attribute. Foreign key (FK) attributes are indicated with the use of FK. The relation formed from the non-repeating attributes will have a foreign key to the relation formed from the repeating attributes. The primary key for the relation for the non-repeating fields will now be a composite key comprising the primary key from the non-repeating relation and the repeating relation.

#### 2NF (2nd Normal Form)

To be in 2nd Normal Form, we must:

* 1. be in 1NF
  2. have no partial dependencies.

Partial dependencies occur when a non-key attribute is only dependent on part of the composite key. If a relation does not have a composite key (that is, the primary key is made up of a single attribute) then it must already be in 2NF.

#### 3NF (3rd Normal Form)

To be in 3rd Normal Form, we must:

1. be in 2NF
2. have no transitive dependencies.

All non-key fields in a relation must be fully functionally dependent on nothing but the primary key. Transitive dependencies occur when a non-key field is dependent on a field other than the primary key.

### Normalisation example

#### Relation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student Number** | **Given name** | **Family name** | **Course** | **Course Name** | **Result** | **Result Description** |
| 10010504 | David | Rossi | MATH1001 | Mathematics 1A | A | Highly Skilled |
| 10010504 | David | Rossi | MATH1002 | Mathematics 1B | B | Skilled |
| 10010504 | David | Rossi | COMP1001 | Computing 1A | A | Highly Skilled |
| 10020423 | James | Stanton | MATH1001 | Mathematics 1A | C | Competent |
| 10020423 | James | Stanton | COMP1001 | Computing 1A | C | Competent |
| 23521461 | Debbie | Tainton | MATH1001 | Mathematics 1A | B | Skilled |
| 23521461 | Debbie | Tainton | MATH1002 | Mathematics 1B | A | Excellent |
| 23521461 | Debbie | Tainton | COMP1001 | Computing 1A | A | Excellent |
| 24352494 | Alison | Roach | MATH1002 | Mathematics 1B | C | Competent |
| 24352494 | Alison | Roach | COMP1001 | Computing 1A | A | Excellent |

This can be written using relational notation:

**Student Results**(Student Number, Given Name, Family Name, Course, Course Name, Results, Result Description)

#### Convert to 1NF

Firstly, check that all attributes are atomic. Then, remove all repeating attributes and place them in another relation.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student Number** | **Given name** | **Family name** |  | **Student Number** | **Course** | **Course Name** | **Result** | **Result Description** |
| 10010504 | David | Rossi |  | 10010504 | MATH1001 | Mathematics 1A | A | Highly Skilled |
| 10020423 | James | Stanton |  | 10010504 | MATH1002 | Mathematics 1B | B | Skilled |
| 23521461 | Debbie | Tainton |  | 10010504 | COMP1001 | Computing 1A | A | Highly Skilled |
| 24352494 | Alison | Roach |  | 10020423 | MATH1001 | Mathematics 1A | C | Competent |
|  |  |  |  | 10020423 | COMP1001 | Computing 1A | C | Competent |
|  |  |  |  | 23521461 | MATH1001 | Mathematics 1A | B | Skilled |
|  |  |  |  | 23521461 | MATH1002 | Mathematics 1B | A | Excellent |
|  |  |  |  | 23521461 | COMP1001 | Computing 1A | A | Excellent |
|  |  |  |  | 24352494 | MATH1002 | Mathematics 1B | C | Competent |
|  |  |  |  | 24352494 | COMP1001 | Computing 1A | A | Excellent |

This can be written using relational notation:

**Student**(Student Number, Given Name, Family Name)

**StudentCourse**(Student Number FK, Course FK, Course Name, Result, Result Description)

#### Convert to 2NF

Check for and remove any partial dependencies. Partial dependencies will only occur in a relation that has a composite key, so Student is already in 2NF.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student Number** | **Given name** | **Family name** |  | **Course** | **Course Name** |  | **Student Number** | **Course** | **Result** | **Result Description** |
| 10010504 | David | Rossi |  | MATH1001 | Mathematics 1A |  | 10010504 | MATH1001 | A | Highly Skilled |
| 10020423 | James | Stanton |  | MATH1002 | Mathematics 1B |  | 10010504 | MATH1002 | B | Skilled |
| 23521461 | Debbie | Tainton |  | COMP1001 | Computing 1A |  | 10010504 | COMP1001 | A | Highly Skilled |
| 24352494 | Alison | Roach |  |  |  |  | 10020423 | MATH1001 | C | Competent |
|  |  |  |  |  |  |  | 10020423 | COMP1001 | C | Competent |
|  |  |  |  |  |  |  | 23521461 | MATH1001 | B | Skilled |
|  |  |  |  |  |  |  | 23521461 | MATH1002 | A | Excellent |
|  |  |  |  |  |  |  | 23521461 | COMP1001 | A | Excellent |
|  |  |  |  |  |  |  | 24352494 | MATH1002 | C | Competent |
|  |  |  |  |  |  |  | 24352494 | COMP1001 | A | Excellent |

This can be written using relational notation:

**Student**(Student Number, Given Name, Family Name)

**Course**(Course, Course Name)

**StudentCourse**(Student Number FK, Course FK, Result, Result Description)

#### Convert to 3NF

Finally, check there are no transitive dependencies. In this case, the result description is dependent on the result, not the course.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student Num** | **Given name** | **Family name** |  | **Course** | **Course Name** |  | **Student Number** | **Course** | **Result** |
| 10010504 | David | Rossi |  | MATH1001 | Mathematics 1A |  | 10010504 | MATH1001 | A |
| 10020423 | James | Stanton |  | MATH1002 | Mathematics 1B |  | 10010504 | MATH1002 | B |
| 23521461 | Debbie | Tainton |  | COMP1001 | Computing 1A |  | 10010504 | COMP1001 | A |
| 24352494 | Alison | Roach |  |  |  |  | 10020423 | MATH1001 | C |
|  |  |  |  |  |  |  | 10020423 | COMP1001 | C |
|  |  |  |  |  |  |  | 23521461 | MATH1001 | B |
| **Result** | **Result Description** | |  |  |  |  | 23521461 | MATH1002 | A |
| A | Highly Skilled | |  |  |  |  | 23521461 | COMP1001 | A |
| B | Skilled | |  |  |  |  | 24352494 | MATH1002 | C |
| C | Competent | |  |  |  |  | 24352494 | COMP1001 | A |

This can be written using relational notation:

**Student**(StudentNum, Given Name, LastName)

**Course**(Course, CourseName)

**StudentCourse**(StudentNum FK, Course FK, Result FK)

**Result**(Result, ResultDescription)

## Common SQL

| **Function** | **SQL syntax** | |
| --- | --- | --- |
| Create table | CREATE TABLE name (  pk INTEGER PRIMARY KEY,  field1 type NOT NULL,  field2 type NULL, …) | |
| Select all data | SELECT \* FROM table | |
| Select specific fields | SELECT field1, field2, field3  FROM table | |
| Select matching rows | SELECT field1, field2  FROM table  WHERE expression | |
| Select data from multiple tables | SELECT table1.field1, table2.field1  FROM table1, table2  WHERE table1.pk = table2.fk | |
| Use aggregate functions | SELECT AVG(field1)  FROM table | |
| Select unique rows | SELECT DISTINCT field1  FROM table | |
| Sort rows | SELECT field1, field2  FROM table  ORDER BY field2 DESC | |
| Group results | SELECT field1, AVG(field2)  FROM table  GROUP BY field1 | |
| Filter grouped results | SELECT field1, AVG(field2)  FROM table  GROUP BY field1  HAVING expression | |
| Concatenate fields | SELECT field1 || field2, field 3  FROM table | |
| Remove table from database | DROP TABLE IF EXISTS table | |
| Insert record into table | INSERT INTO table (field1, field2)  VALUES (value1, value2) | |
| Delete all records from table | DELETE FROM table | |
| Delete specific records from table | DELETE FROM table  WHERE condition | |
| Change records in a table | UPDATE table  SET field1 = value  WHERE expression | |
| Comparison operators | = | Equal to |
| <> or != | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| Logic operators | ALL | returns TRUE if all expressions are TRUE. |
| AND | returns TRUE if both expressions are TRUE, and FALSE if one of the expressions is FALSE. |
| ANY | returns TRUE if any one of a set of comparisons is TRUE. |
| BETWEEN | returns TRUE if a value is within a range. |
| EXISTS | returns TRUE if a subquery contains any rows. |
| IN | returns TRUE if a value is in a list of values. |
| LIKE | returns TRUE if a value matches a pattern (use with the wildcard characters % and \_) |
| NOT | reverses the value of other operators such as NOT EXISTS, NOT IN, NOT BETWEEN, etc. |
| OR | returns TRUE if either expression is TRUE |
| Aggregate functions | AVG | calculate the average value |
| COUNT | count the number of items in a set |
| MAX | find the maximum value |
| MIN | find the minimum value |
| SUM | calculate the sum of values |

# Appendices

Python is the prescribed programming language for the Computer Science ATAR course and will be used in ATAR examination questions related to programming.

## Control Structure Python Examples

| **Pseudocode** | **Python** |
| --- | --- |
| INPUT(num1)  INPUT(num2)  product = num1 \* num2  PRINT(product) | #sequence  num1 = int(input("First num: "))  num2 = int(input("Second num: "))  product = num1 \* num2  print(product) |
| speed = 50  IF speed > 50 THEN  PRINT(“You are speeding”)  END IF | #selection - IF  speed = 50  if speed > 50:  print("You are speeding") |
| speed = 50  IF speed > 50 THEN  PRINT(“You are speeding”)  ELSE  PRINT(‘You are not speeding”)  END IF | #selection - IF ELSE  speed = 50  if speed > 50:  print("You are speeding")  else:  print("You are not speeding") |
| **Method 1 – IF…ELSE IF…ELSE**  speed = 50  IF speed < 20 THEN  PRINT(“You are going too slow”)  ELSE IF speed > 50 THEN  PRINT(“You are speeding”)  ELSE  PRINT(“You are not speeding”)  END IF | #selection - IF ELIF ELSE  speed = 50  if speed < 20:  print("You are going too slow")  elif speed > 50:  print("You are speeding")  else:  print("You are not speeding") |
| **Method 2 – CASE statement**  colour = ‘red’  CASE colour OF  ’red’: PRINT(“Stop”)  ‘yellow’: PRINT (“Slow down”)  ‘green’: PRINT(“Go”)  OTHER: PRINT(“Incorrect colour”)  END CASE | #selection CASE (match in Python)  colour = "red"  match colour:  case "red":  print("Stop")  case "yellow":  print("Slow down")  case "green":  print("Go")  case other:  print("Incorrect colour") |
| num = 0  WHILE num < 10  PRINT(“The number is “ + num)  num = num + 1  END WHILE | #Test first loop (while)  num = 0  while num < 10:  print("The number is {num}")  num = num + 1 |
| REPEAT  INPUT(Age)  UNTIL (Age >= 6) AND (Age <= 17)  PRINT (Age) | #Test last loop (repeat until)  #No structure exists to natively implement this in Python, but this is functionally identical  age = input("Age: ")  while age < 6 and age > 17:  age = input("Age: ")  print(age) |
| FOR num = 1 TO 10  PRINT(“The number is “ + num)  END FOR  FOR num = 10 TO 1 STEP –1  PRINT(num)  END FOR  PRINT("Blast off!”)  FOR num = 1 TO 100 STEP 10  PRINT(“The number is “ + num)  END FOR | #Fixed loops - FOR  for num in range(1,11):  print("The number is: {num}")  for num in range(10,0, -1):  print(num)  print("Blast off!")  for num in range(1,100,10):  print("The number is {num}") |

## Object Oriented Python Examples

class Animal:

name = ""

hunger = 5

food\_list = []

#Functions named "\_\_init\_\_" act as constructors in Python

def \_\_init\_\_(self, new\_name):

name = new\_name

def eat(self, food):

result = ""

if food in self.food\_list:

result = "Not hungry"

if hunger > 0:

self.hunger = self.hunger - 1

result = "That was yummy"

else:

result = "I don't like that food"

return result

def is\_hungry(self):

return self.hunger > 0

horse = Animal("Silver") #Creates a horse with the name "Silver"

horse.food\_list.append("grass") #Will add grass to the food\_list

horse.eat("potato") #Will return “I don’t like that food”

#To indicate inheritance in Python, the class will receive the parent as a parameter

class Dog(Animal):

has\_fur = True

legs = 4

food\_list = ["meat", "bones"]

def bark(self):

return f"{self.name} is barking"

def number\_of\_legs(self):

return self.legs

class Fish(Animal):

has\_fins = True

food\_list = ["algae", "plankton"]

def swim(self):

return f"{self.name} is swimming"

fido = Dog("Fido")

print(fido.number\_of\_legs())

goldie = Fish("Goldie")

print(goldie.has\_fins)

**Array Examples**

#Load an array

def LoadArray():

  name = ""

  i = 0

  names = []

  name = input("Enter a name: ")

  while name != "":

    names.append(name)

    i = i + 1

    name = input("Enter a name: ")

  print(f"There were {i} names entered.")

#Print contents of an array

def PrintArray():

  names = ["Peter", "Jane", "Hugo", "Kai", "Sally", "Arman"]

  for i in range(len(names)):

    print(names[i])

#Add contents of an array

def AddArray():

  numbers = [4, 8, 23, 52, 3, 27, 86]

  total = 0

  for i in range(len(numbers)):

    total = total + numbers[i]

  print(total)

#Minimum value in array

def FindMinimumValue():

  numbers = [4, 8, 23, 52, 3, 27, 86]

  min = numbers[0]

  minIndex = 0

  for i in range(len(numbers)):

    if numbers[i] < min:

      min = numbers[i]

      minIndex = i

  print(f"The minimum value is {min}")

  print(f"The minimum value is at position {minIndex}")

#Maximum value in array

def FindMaximumValue():

  numbers = [4, 8, 23, 52, 3, 27, 86]

  max = numbers[0]

  maxIndex = 0

  for i in range(len(numbers)):

    if numbers[i] > max:

      max = numbers[i]

      maxIndex = i

  print(f"The maximum value is {max}")

  print(f"The maximum value is at position {maxIndex}")

**File Processing**

#Note that Python has several methods to open and access files

#These examples have been created to most closely match the provided pseudocode

def ReadFile():

  myfile = open("data.txt")

  lines = []

  line = myfile.readline()

  while line != "":

    lines.append(line.strip())

    line = myfile.readline()

  myfile.close()

def WriteFile():

  newline = "\n"

  myfile = open("outputfile.txt", "w")

  lines = ["Twinkle Twinkle Little Star", "Baa Baa Black Sheep", "Hickory Dickory Dock"]

  for i in range(len(lines)):

    myfile.write(lines[i] + newline)

  myfile.close()

def AppendFile():

  newline = "\n"

  myfile = open("names\_file.txt", "a")

  names = ["James Smith", "Aaron Jones", "Sally Gonzales"]

  for i in range(len(names)):

    myfile.write(names[i] + newline)

  myfile.close()

**Search Algorithms**

def LinearSearch(searchArray, target):

    index = 0

    position = -1

    while index < len(searchArray) and position == -1:

      if searchArray[index] == target:

        position = index

      index = index + 1

    return position

def BinarySearch(searchArray, target):

    position = -1

    lowerBound = 0

    upperBound = len(searchArray) - 1

    while lowerBound <= upperBound and position == -1:

        midpoint = (lowerBound + upperBound) // 2

        if searchArray[midpoint] < target:

            lowerBound = midpoint + 1

        elif searchArray[midpoint] > target:

            upperBound = midpoint - 1

        else:

          position = midpoint

    return position

**Sort Algorithms**

def BubbleSort(arrayToSort):

    last = len(arrayToSort) - 1

    swapped = True

    while swapped:

        swapped = False

        i = 0

        while i < last:

            if arrayToSort[i] > arrayToSort[i + 1]:

                temp = arrayToSort[i]

                arrayToSort[i] = arrayToSort[i + 1]

                arrayToSort[i + 1] = temp

                swapped = True

            i = i + 1

        last = last - 1

    return(arrayToSort)

def BubbleSort(arrayToSort):

    last = len(arrayToSort) - 1

    swapped = True

    while swapped:

        swapped = False

        i = 0

        while i < last:

            if arrayToSort[i] > arrayToSort[i + 1]:

                temp = arrayToSort[i]

                arrayToSort[i] = arrayToSort[i + 1]

                arrayToSort[i + 1] = temp

                swapped = True

            i = i + 1

        last = last - 1

    return(arrayToSort)

def InsertionSort(arrayToSort):

    position = 0

    while position < len(arrayToSort):

        currentValue = arrayToSort[position]

        sortedPosition = position - 1

        while sortedPosition >= 0 and arrayToSort[sortedPosition] > currentValue:

            arrayToSort[sortedPosition + 1] = arrayToSort[sortedPosition]

            sortedPosition = sortedPosition - 1

        arrayToSort[sortedPosition + 1] = currentValue

        position = position + 1

    return arrayToSort

def SelectionSort(arrayToSort):

    unsortedIndex = len(arrayToSort) - 1

    while unsortedIndex > 0:

        i = 0

        max = arrayToSort[i]

        maxIndex = i

        while i < unsortedIndex:

            i = i + 1

            if arrayToSort[i] > max:

                max = arrayToSort[i]

                maxIndex = i

        temp = arrayToSort[maxIndex]

        arrayToSort[maxIndex] = arrayToSort[unsortedIndex]

        arrayToSort[unsortedIndex] = temp

        unsortedIndex = unsortedIndex - 1

    return arrayToSort