



Government of **Western Australia**
School Curriculum and Standards Authority

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](#).

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. Teachers must exercise their professional judgement as to the appropriateness of any they may wish to use.

Contents

Purpose	1
Programming	1
Conventions for writing pseudocode	1
Modularisation	4
Parameters	5
Object-oriented programming	5
Creating a new class	5
Instantiating and using an object	6
Inheritance	6
Common algorithms	7
Search algorithms	9
Sort algorithms	10
Network communications	11
Key protocols associated with layers in models	11
Network diagram conventions (CISCO)	12
Subnetting	13
Cyber security	14
Common methods of encryption	14
Data management	15
Entity relationship diagrams	15
Data dictionaries	16
Normalisation	16
Common SQL	22
Appendix 1: Python control structures	24
Appendix 2: Python object-oriented programming examples	26
Array examples	27
File processing	28
Search algorithms	29
Sort algorithms	29
Acknowledgements	31

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 Year 11 and Year 12 Computer Science ATAR courses and will be used in ATAR examination questions related to programming.

Conventions for writing pseudocode

- Pseudocode is used in this course to express various algorithms. There is no specific syntax required although algorithms should be clear and easy to follow. The following conventions have been provided as guidance only. 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 and use upper case.
- Global variables should be initialised outside the mainline function.

Table 1. 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

Command	Pseudocode
AND	<code>x > 1 AND x < 10</code>
Arrays	<pre>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 two_d_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Create a two-dimensional array (list of lists) third_row = two_d_array[2] # Returns [7, 8, 9] second_element_third_row = two_d_array[1][2] # Returns 6</pre>
Dictionaries	<pre>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 # array of all keys in the dictionary costOfGear.values # array of all values in the dictionary costOfGear.items # array of all key/value pairs in the dictionary</pre>

Table 2. Programming control structures

Structure	Example
Sequence	<pre>INPUT(num1) INPUT(num2) product = num1 * num2 PRINT(product)</pre>
One-way selection	
<pre>IF condition THEN do something END IF</pre>	<pre>IF speed > 50 THEN PRINT("You are speeding") END IF</pre>
Two-way selection	
<pre>IF condition THEN do something ELSE do something END IF</pre>	<pre>IF speed > 50 THEN PRINT("You are speeding") ELSE PRINT("You are not speeding") END IF</pre>
Multi-way selection (IF ... ELSE)	
<pre>IF condition THEN</pre>	<pre>IF speed < 20 THEN</pre>

Structure	Example
<pre> do something ELSE IF condition THEN do something ELSE do something END IF </pre>	<pre> PRINT("You are going too slow") ELSE IF speed > 50 THEN PRINT("You are speeding") ELSE PRINT("You are not speeding") END IF </pre>
Multi-way selection (CASE)	
<pre> CASE value OF choice 1: do something choice 2: do something OTHERWISE: do something END CASE </pre>	<pre> CASE colour OF 'red': PRINT("Stop") 'yellow': PRINT("Slow down") 'green': PRINT("Go") OTHER: PRINT("Incorrect colour") END CASE </pre> <p>Note: CASE statements should only use single values and not evaluate ranges. When evaluating a range of values, use IF statements.</p>
Test-first loop (WHILE)	
<pre> WHILE condition is TRUE do something END WHILE </pre>	<pre> num = 0 WHILE num < 10 PRINT("The number is ", num) num = num + 1 END WHILE </pre>
Test-last loop (REPEAT UNTIL)	
<pre> REPEAT do something UNTIL condition is TRUE </pre>	<pre> REPEAT INPUT(age) UNTIL (age >= 6) AND (age <= 17) PRINT (age) </pre>
Fixed loop (FOR)	
<pre> FOR variable = start TO finish [STEP increment] do something END FOR </pre>	<pre> 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 </pre>

Modularisation

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

- allowing code to be reused and reducing code repetition
- allowing more people to work on a project – each person can work on separate modules
- breaking down large complex problem into smaller problems to make it easier to solve
- making algorithms and programs easier to read
- making errors quicker and easier to find.

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.

In the table below, the code in the left-hand column repeats the same lines of code three times where it calculates the area based on the length and height. The code in the right-hand column reduces this repetition by moving those lines of code to a separate module.

Table 3. Modularisation comparison table

Without modularisation	With modularisation
<pre> 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 </pre>	<pre> 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 </pre>

Parameters

Parameters are used 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 objects) will be passed by reference.

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 a 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 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

Iterating over an array:

There are two main methods for iterating over an array – looping through the elements of the array or using the index. Depending on the use-case, both methods may be acceptable.

```
names = ["Peter", "Jane", "Hugo", "Kai", "Sally", "Arman"]
FOR name IN names
    PRINT(name)
END FOR

FOR i = 0 TO names.length - 1
    PRINT(names[i])
END FOR
```

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]
    min_index = 0
    FOR i = 1 TO numbers.length - 1
        IF numbers[i] < min THEN
            min = numbers[i]
            min_index = i
        END IF
    END FOR
    PRINT("The minimum value is", min)
    PRINT("The minimum value is at position", min_index)
END FindMinimumValue

```

Maximum value in array:

```

FUNCTION FindMaximumValue
    numbers = [4, 8, 23, 52, 3, 27, 86]
    max = numbers[0]
    max_index = 0
    FOR i = 1 TO numbers.length - 1
        IF numbers[i] > max THEN
            max = numbers[i]
            max_index = i
        END IF
    END FOR
    PRINT("The maximum value is ", max)
    PRINT("The maximum value is at position ", max_index)
END FindMaximumValue

```

File processing

```

FUNCTION ReadFile
    myfile = OPEN_READ("data.txt")
    lines = []
    WHILE NOT myfile.end_of_file
        line = myfile.READLINE()
        lines.append(line)
    END WHILE
    CLOSE(myfile)
END ReadFile

FUNCTION WriteFile
    myfile = OPEN_WRITE("outputfile.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 AppendFile

```

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(search_array, target)
        index = 0
        position = -1
        WHILE index < search_array.length AND position == -1
            IF search_array[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. It compares the target with a value from the middle of the array. If not a match, it uses the comparison to decide which remaining half of the array to search and repeats the process, 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(search_array, target)
        position = -1
        lower_bound = 0
        upper_bound = search_array.length - 1
        WHILE lower_bound <= upper_bound AND position == -1
            midpoint = (lower_bound + upper_bound) / 2
            IF search_array[midpoint] < target THEN

```

```

        lowerBound = midpoint + 1
    ELSE IF search_array[midpoint] > target THEN
        upper_bound = midpoint - 1
    ELSE
        position = midpoint
    END IF
END WHILE
RETURN position
END BinarySearch

```

Sort algorithms

Bubble sort

```

FUNCTION BubbleSort(array_to_sort)
    last = array_to_sort.length - 1
    swapped = TRUE
    WHILE swapped
        swapped = FALSE
        i = 0
        WHILE i < last
            IF array_to_sort[i] > array_to_sort[i + 1] THEN
                temp = array_to_sort[i]
                array_to_sort[i] = array_to_sort[i + 1]
                array_to_sort[i + 1] = temp
                swapped = TRUE
            END IF
            i = i + 1
        END WHILE
        last = last - 1
    END WHILE
    RETURN array_to_sort
END BubbleSort

```

Insertion sort

```

FUNCTION InsertionSort(array_to_sort)
    position = 0
    WHILE position < array_to_sort.length
        Current_value = array_to_sort[position]
        Sorted_position = position - 1
        WHILE sorted_position >= 0 and array_to_sort[sorted_position] >
current_value
            Array_to_sort[sorted_position + 1] = array_to_sort[sorted_position]
            Sorted_position = sorted_position - 1
        END WHILE
        Array_to_sort[sorted_position + 1] = current_value
    END WHILE

```

```

        position = position + 1
    END WHILE
    return array_to_sort
END InsertionSort

```

Selection sort

```

FUNCTION SelectionSort(array_to_sort)
    unsorted_index = array_to_sort.length - 1
    WHILE unsorted_index > 0
        i = 0
        max = array_to_sort[i]
        max_index = i
        WHILE i <= unsorted_index
            i = i + 1
            IF array_to_sort[i] > max THEN
                max = array_to_sort[i]
                max_index = i
            END IF
        END WHILE
        temp = array_to_sort[max_index]
        array_to_sort[max_index] = array_to_sort[unsorted_index]
        array_to_sort[unsorted_index] = temp
        unsorted_index = unsorted_index - 1
    END WHILE
    RETURN array_to_sort
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.

Table 4. Examples of key protocols from the DoD TCP/IP model

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

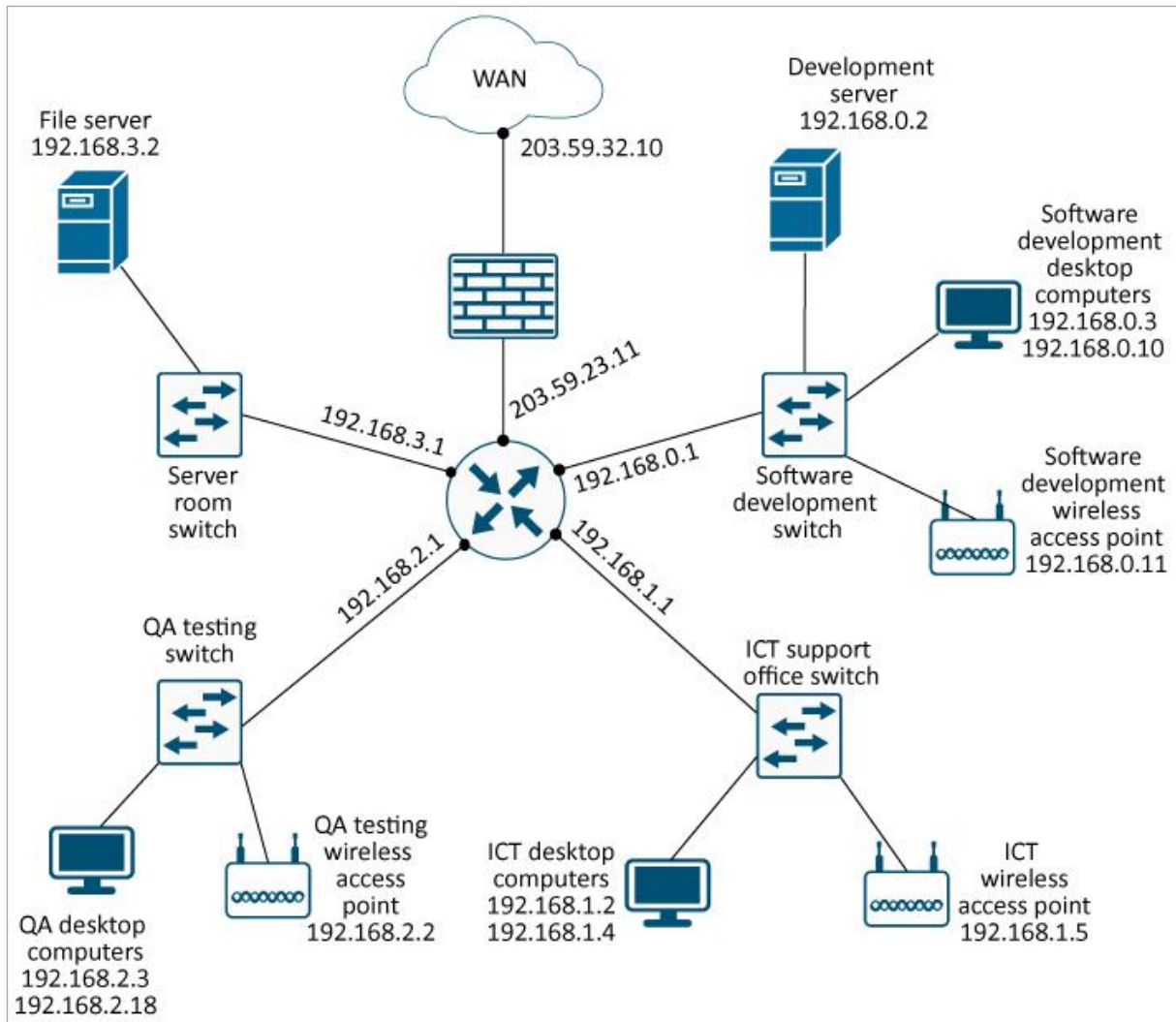
Network diagram conventions (CISCO)

Table 5. Network diagram conventions

Network component	Symbol
WAN/internet/internet service provider (ISP)	
Router	
Modem	
Switch	
Wireless access point	
End devices	
File server	
Database server	
Firewall	

For the purposes of the Years 11 and 12 ATAR syllabuses, firewalls should be placed outside the trusted network and do not require an IP address.

Figure 1. Network diagram example



Subnetting

Subnetting is a technique used in networking to divide a larger IP address block into smaller segments called subnets. This helps in efficiently managing IP addresses, improving network performance and enhancing security. A subnet mask determines which part of the IP address is for the network and which is for the host. By 'borrowing' bits from the host portion of an IP address, subnets are created with specific ranges of addresses. Subnetting allows for better IP address management, reduces network congestion and enhances security by isolating network segments.

For example, to divide the network 192.168.1.0/24 into two subnets, you can borrow 1 bit from the host portion, changing the subnet mask from 255.255.255.0 to 255.255.255.128.

Original network: 192.168.1.0/24 (subnet mask: 255.255.255.0)

Host range: 192.168.1.1 to 192.168.1.254

New subnet 1: 192.168.1.0/25 (subnet mask: 255.255.255.128)

Host range: 192.168.1.1 to 192.168.1.126

New subnet 2: 192.168.1.128/25 (subnet mask: 255.255.255.128)

Host range: 192.168.1.129 to 192.168.1.254

Table 6. IPv4 classes and default subnet masks

	IP address ranges	Network and host	Default subnet mask
Class A	10.0.0.0 - 10.255.255.255	Network.Host.Host.Host	255.0.0.0
Class B	172.16.0.0 - 172.31.255.255	Network.Network.Host.Host	255.255.0.0
Class C	192.168.0.0 - 192.168.255.255	Network.Network.Network.Host	255.255.255.0

Cyber security

Common methods of encryption

Early methods and weaknesses

- **Substitution cipher** swaps out characters. Assuming 26 alphabet characters, this form of encryption is easily broken using character frequency.
- **Vigenère cipher** uses a repeated key, combining plain text with the key. This form of encryption is easily broken if the length of the key is known and the character frequency method is used, 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 that used a key size of 56 bits. That is small compared to today's standards and can be quickly cracked with fast processing speeds.
- **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 when trying to securely communicate with someone who does not have the 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 slow compared to AES, so is often used to securely communicate the private AES key. RSA uses 2048–4096 key sizes and works using a key produced by an algorithm from two prime numbers.
- **Elliptical curve cryptography (ECC)** has been introduced more recently as an alternative asymmetric encryption algorithm to RSA, requiring smaller keys and shorter encryption times.

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.
- Users should encrypt any messages and critical or sensitive files they send. This extends to the encryption of storage devices.

Best practice is based on the guidelines from the National Institute of Standards and Technology (NIST) available at <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.

Figure 2. Example of an entity relationship diagram

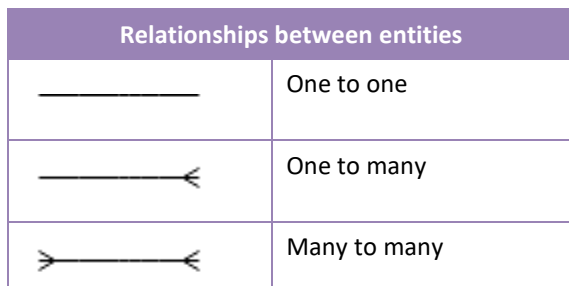
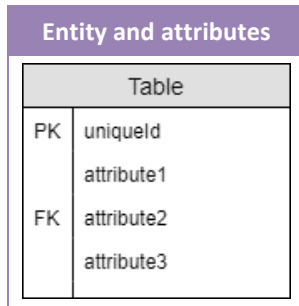
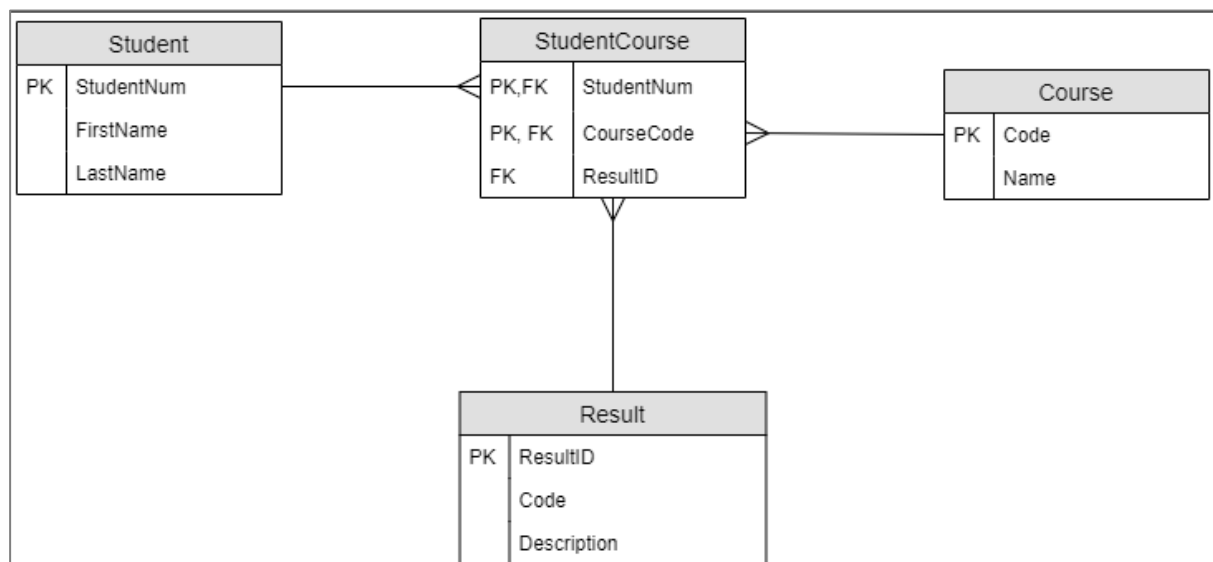


Figure 3. Example of an entity relationship diagram



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.

Table 7. Data dictionary example

ElementName	DataType	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: the description should be brief and include information about the data being stored, the format of the data and the default value if applicable.

Normalisation

Normalisation is the process of eliminating data dependencies and redundancies to improve data integrity and efficiency in a relational database, and reducing the likelihood of anomalies. 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.

StudentNumber	GivenName	FamilyName	Email	Course	CourseName
10010504	David	Rossi	drossi@student.edu.au	MATH1001	Mathematics 1A
10010504	David	Rossi	drossi@student.edu.au	COMP1001	Computing 1A
10010504	David	Rossi	drossi@student.edu.au	MATH1002	Mathematics 1B
24352494	Debbie	Tainton	dtainton@student.edu.au	MATH1002	Mathematics 1B
24352494	Alison	Roach	aroach@student.edu.au	MATH2001	Mathematics 2B

Update anomaly

An update anomaly occurs when the data being updated 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 their email address, then all three occurrences need to be updated.

Delete anomaly

A delete anomaly occurs when one piece of data is deleted, which results in the only instance of another piece of data being deleted. For example, if Alison Roach was removed from the database, then information about the subject Mathematics 2B would also be lost.

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 is allocated, then a new record cannot be created for the subject, as necessary information is missing.

Normalisation to third normal form (3NF)

Steps to normalisation of data:

1. ensure data is in the form of a relation
2. convert data to first normal form (1NF)
3. convert data to second normal form (2NF)
4. convert data to third normal form (3NF).

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.

StudentNumber	GivenName	FamilyName	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.

StudentNumber	GivenName	FamilyName	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 the process of normalisation can begin.

Table 8. Relation example

StudentNumber	GivenName	FamilyName	Email	Course	CourseName
10010504	David	Rossi	<u>drossi@student.edu.au</u>	MATH1001	Mathematics 1A
10010504	David	Rossi	<u>drossi@student.edu.au</u>	COMP1001	Computing 1A
10010504	David	Rossi	<u>drossi@student.edu.au</u>	MATH1002	Mathematics 1B
24352494	Debbie	Tainton	<u>dtainton@student.edu.au</u>	MATH1002	Mathematics 1B
24352494	Alison	Roach	<u>aroach@student.edu.au</u>	MATH2001	Mathematics 2B

Process of normalisation

1NF

To be in 1NF:

- all fields must be atomic
- all repeating attributes must be removed
- each record must be unique with a primary key.

Each relation that is formed will have a primary key. Primary keys are indicated via underlining of 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

To be in 2NF:

- the data must already be in 1NF
- there must be no partial dependencies.

Partial dependencies occur when a non-key attribute is dependent only 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

To be in 3NF:

- the data must already be in 2NF
- there must be 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

StudentNumber	GivenName	FamilyName	Course	CourseName	Result	ResultDescription
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(StudentNumber, GivenName, FamilyName, Course, CourseName, Results, ResultDescription)

Convert to 1NF

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

StudentNumber	GivenName	FamilyName
10010504	David	Rossi
10020423	James	Stanton
23521461	Debbie	Tainton
24352494	Alison	Roach

StudentNumber	Course	CourseName	Result	ResultDescription
10010504	MATH1001	Mathematics 1A	A	Highly Skilled
10010504	MATH1002	Mathematics 1B	B	Skilled
10010504	COMP1001	Computing 1A	A	Highly Skilled
10020423	MATH1001	Mathematics 1A	C	Competent
10020423	COMP1001	Computing 1A	C	Competent
23521461	MATH1001	Mathematics 1A	B	Skilled
23521461	MATH1002	Mathematics 1B	A	Excellent

StudentNumber	Course	CourseName	Result	ResultDescription
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(StudentNumber, GivenName, FamilyName)

StudentCourse(StudentNumber FK, Course FK, CourseName, Result, ResultDescription)

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.

StudentNumber	GivenName	FamilyName
10010504	David	Rossi
10020423	James	Stanton
23521461	Debbie	Tainton
24352494	Alison	Roach

Course	CourseName
MATH1001	Mathematics 1A
MATH1002	Mathematics 1B
COMP1001	Computing 1A

StudentNumber	Course	Result	ResultDescription
10010504	MATH1001	A	Highly Skilled
10010504	MATH1002	B	Skilled
10010504	COMP1001	A	Highly Skilled
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(StudentNumber, GivenName, FamilyName)

Course(Course, CourseName)

StudentCourse(StudentNumber FK, Course FK, Result, ResultDescription)

Convert to 3NF

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

StudentNumber	GivenName	FamilyName
10010504	David	Rossi
10020423	James	Stanton
23521461	Debbie	Tainton
24352494	Alison	Roach

Course	CourseName
MATH1001	Mathematics 1A
MATH1002	Mathematics 1B
COMP1001	Computing 1A

StudentNumber	Course	Result
10010504	MATH1001	A
10010504	MATH1002	B
10010504	COMP1001	A
10020423	MATH1001	C
10020423	COMP1001	C
23521461	MATH1001	B
23521461	MATH1002	A
23521461	COMP1001	A
24352494	MATH1002	C
24352494	COMP1001	A

Result	ResultDescription
A	Highly Skilled
B	Skilled
C	Competent

This can be written using relational notation:

Student(StudentNum, GivenName, LastName)

Course(Course, CourseName)

StudentCourse(StudentNum FK, Course FK, Result FK)

Result(Result, ResultDescription)

Common SQL

Table 9. Common functions and SQL syntax

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

Function	SQL syntax	
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, or FALSE if any 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

Appendix 1: Python control structures

Python is the prescribed programming language for the Year 11 and Year 12 Computer Science ATAR courses and will be used in ATAR examination questions related to programming.

Table 10. Python control structures examples

Pseudocode	Python
INPUT(num1) INPUT(num2) product = num1 * num2 PRINT(product)	<pre>#sequence num1 = int(input("First num: ")) num2 = int(input("Second num: ")) product = num1 * num2 print(product)</pre>
IF speed > 50 THEN PRINT("You are speeding") END IF	<pre>#selection - IF if speed > 50: print("You are speeding")</pre>
IF speed > 50 THEN PRINT("You are speeding") ELSE PRINT("You are not speeding") END IF	<pre>#selection - IF ELSE if speed > 50: print("You are speeding") else: print("You are not speeding")</pre>
Method 1 – IF...ELSE IF...ELSE 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	<pre>#selection - IF ELIF ELSE if speed < 20: print("You are going too slow") elif speed > 50: print("You are speeding") else: print("You are not speeding")</pre>

Pseudocode	Python
Method 2 – CASE statement CASE colour OF 'red': PRINT("Stop") 'yellow': PRINT("Slow down") 'green': PRINT("Go") OTHER: PRINT("Incorrect colour") END CASE	#selection CASE (match in Python) 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}")

Appendix 2: Python object-oriented programming 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]
    min_index = 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 {min_index}")
#Maximum value in array
def FindMaximumValue():
    numbers = [4, 8, 23, 52, 3, 27, 86]
    max = numbers[0]
    max_index = 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 {max_index}")
```

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(search_array, target):
    index = 0
    position = -1
    while index < len(search_array) and position == -1:
        if search_array[index] == target:
            position = index
            index = index + 1
    return position

def BinarySearch(search_array, target):
    position = -1
    lower_bound = 0
    upper_bound = len(search_array) - 1
    while lower_bound <= upper_bound and position == -1:
        midpoint = (lower_bound + upper_bound) // 2
        if search_array[midpoint] < target:
            lower_bound = midpoint + 1
        elif search_array[midpoint] > target:
            upper_bound = midpoint - 1
        else:
            position = midpoint
    return position
```

Sort algorithms

```
def BubbleSort(array_to_sort):
    last = len(array_to_sort) - 1
    swapped = True
    while swapped:
        swapped = False
        i = 0
        while i < last:
```

```
        if array_to_sort[i] > array_to_sort[i + 1]:
            temp = array_to_sort[i]
            array_to_sort[i] = array_to_sort[i + 1]
            array_to_sort[i + 1] = temp
            swapped = True
        i = i + 1
    last = last - 1
    return(array_to_sort)

def InsertionSort(array_to_sort):
    position = 0
    while position < len(array_to_sort):
        current_value = array_to_sort[position]
        sorted_position = position - 1
        while sorted_position >= 0 and array_to_sort[sorted_position] >
current_value:
            array_to_sort[sorted_position + 1] =
array_to_sort[sorted_position]
            sorted_position = sorted_position - 1
        array_to_sort[sorted_position + 1] = current_value
        position = position + 1
    return array_to_sort

def SelectionSort(array_to_sort):
    unsorted_index = len(array_to_sort) - 1
    while unsorted_index > 0:
        i = 0
        max = array_to_sort[i]
        xindex = i
        while i <= unsorted_index:
            if array_to_sort[i] > max:
                max = array_to_sort[i]
                max_index = i
            i = i + 1
        temp = array_to_sort[max_index]
        array_to_sort[max_index] = array_to_sort[unsorted_index]
        array_to_sort[unsorted_index] = temp
        unsorted_index = unsorted_index - 1
    return array_to_sort
```

Acknowledgements

Cisco Systems Inc. *Network Topology Icons*. Retrieved September, 2024, from <https://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html>