OCR Computer Science Syllabus and Notes

1b. Aims and learning outcomes OCR's GCSE (9–1)

Computer Science will encourage students to:

• understand and apply the fundamental principles and concepts of Computer Science, including abstraction, decomposition, logic, algorithms, and data representation

• analyse problems in computational terms through practical experience of solving such problems, including designing, writing and debugging programs

• think creatively, innovatively, analytically, logically and critically

• understand the components that make up digital systems, and how they communicate with one another and with other systems

• understand the impacts of digital technology to the individual and to wider society • apply mathematical skills relevant to Computer Science.

J277/01: Computer systems [exam paper 1]

Topic	Where in Program of Study
1.1 Systems architecture	Early Y10 (Hardware topic)
1.2 Memory and storage	Early Y10 (Hardware topic)
1.3 Computer networks, connections and protocols	Late Y10 (Networks)
1.4 Network security	Late Y10 (Networks)
1.5 Systems software	Mid Y10 (Operating systems and Apps)
1.6 Ethical, legal, cultural and environmental impacts of digital technology	Early Y11

J277/02: Computational thinking, algorithms and programming [exam paper 2]

Topic	Where in Program of Study
2.1 Algorithms	Early Y10 (Hardware topic)
2.2 Programming fundamentals	Early Y10 (Hardware topic)
2.3 Producing robust programs	Late Y10 (Networks)
2.4 Boolean logic	Mid Y10 (Logic gates)
2.5 Programming languages and Integrated Development Environments	Early Y11

2b. Content of Computer systems (J277/01)

1.1 Systems architecture

Sub topic and details	Guidance and Notes
1.1.1 Architecture of the CPU The purpose of the CPU: Alex_Cheung_CPU_pages o The fetch-execute cycle **CPU components and function:** o ALU (Arithmetic Logic Unit) [calculations, comparisons, bit manipulations etc.] o CU (Control Unit) o Cache o Registers **Von Neumann architecture:** o MAR (Memory Address Register) o MDR (Memory Data Register) o Program Counter o Accumulator	What actions occur at each stage of the fetch-execute cycle – The role/purpose of each component and what it manages, stores, or controls during the fetch-execute cycle. The purpose of each register, what it stores (data or address). The difference between storing data and an address. The classic Von Neuman machine instruction cycle: FETCH: The address of the next instruction to be fetched is transferred from the PC to the MAR and the instruction is fetched from memory into the MDR and stored in another register ([C]IR); the PC is incremented. DECODE: The CU decodes the instruction into signals for the other components EXECUTE: The ALU will execute any arithmetic or logical operations and the MAR and MDR will be used for any data transfer operations STORE: The result of the operations performed is stored in the Acc. (Not required - Knowledge of passing of data between registers in each stage)
1.1.2 CPU performance How common characteristics of CPUs affect their performance: o Clock speed o Cache size o Number of cores	Understanding of each characteristic as listed. The effects of changing any of the common characteristics on system performance, either individually or in combination. A faster clock speed will increase performance for all applications unless there is another factor limiting the performance such as GPU speed or secondary storage access times. A larger cache size will increase performance as frequently and recently used data for active tasks can be held in Cache and access in one CPU cycle. More cores will improve performance for software that is programmed to take advantage of multi-threading.
1.1.3 Embedded systems **The purpose and characteristics of embedded systems** Examples of embedded systems	What embedded systems are. Typical characteristics of embedded systems. Familiarity with a range of different embedded systems (eg the computer in a washing machine)

1.2 – Memory and storage	
Sub topic and details	Guidance and Notes
1.2.1 Primary storage (Memory) ¨ The need for primary storage ¨ The difference between RAM and ROM ¨ The purpose of ROM in a computer system ¨ The purpose of RAM in a computer system ¨ Virtual memory	Computers have primary storage to enable instructions and data to be addressed (accessed directly) by the CPU for processing. Primary storage usually consists of RAM and ROM Key characteristics of RAM and ROM Why virtual memory may be needed in a system How virtual memory works - Transfer of memory pages between RAM and HDD when RAM is filled
1.2.2 Secondary storage ¨ The need for secondary storage ¨ Common types of storage: o Optical o Magnetic o Solid state ¨ Suitable storage devices and storage media for a given application ¨ The advantages and disadvantages of different storage devices and storage media relating to these characteristics: o Capacity o Speed o Portability o Durability o Reliability o Cost	Why computers have secondary storage. Recognise a range of secondary storage devices/media. Differences between each type of storage device/medium. Compare advantages/disadvantages for each storage device. Be able to apply their knowledge in context within scenarios. Not required Understanding of the component parts of these types of storage.
1.2.3 Units	

The units of data storage: o Bit o Nibble (4 bits) o Byte (8 bits) o Kilobyte (1,000 bytes or 1 KB) o Megabyte (1,000 KB) o Gigabyte (1,000 MB) o Terabyte (1,000 GB) o Petabyte (1,000 TB) ¨ How data needs to be converted into a binary format to be processed by a computer ¨ Data capacity and calculation of data capacity requirements	Why data must be stored in binary format (It is the simplest form of data storage and doesn't need complex hardware to read the data) (the simplicity of binary also allows the computer to read the information quicker, and transfer information as electrical signals to the transistors which can only work with binary) Familiarity with data units and moving between each Data storage devices have different fixed capacities Calculate required storage capacity for a given set of files Calculate file sizes of sound, images and text files § sound file size = sample rate x duration (s) x bit depth § image file size = colour depth x image height (px) x image width (px) § text file size = bits per character x number of characters Alternatives • Use of 1,024 for conversions and calculations would be acceptable • Allowance for metadata in calculations may be used
1.2.4 Data storage Numbers ¨ How to convert positive denary whole numbers to binary numbers (up to and including 8 bits) and vice versa ¨ How to add two binary integers together (up to and including 8 bits) and explain overflow errors which may occur ¨ How to convert positive denary whole numbers into 2-digit hexadecimal numbers and vice versa	Denary number range 0 – 255 Hexadecimal range 00 – FF Binary number range 00000000 – 11111111 Understanding of the terms 'most significant bit', and 'least significant bit' Conversion of any number in these ranges to another number base Ability to deal with binary numbers containing between 1 and 8 bits .g. 11010 is the same as 00011010 Understand the effect of a binary shift (both left or right) on a number

¨ How to convert binary integers to their hexadecimal equivalents and vice versa ¨ Binary shifts	Carry out a binary shift (both left and right)
Characters ¨ The use of binary codes to represent characters ¨ The term 'character set' ¨ The relationship between the number of bits per character in a character set, and the number of characters which can be represented, e.g.: o ASCII o Unicode	How characters are represented in binary How the number of characters stored is limited by the bits available The differences between and impact of each character sets Understand how character sets are logically ordered, e.g. the code for 'B' will be one more than the code for 'A' Binary representation of ASCII in the exam will use 8 bits (Not required - Memorisation of character set codes)
Images ¨ How an image is represented as a series of pixels, represented in Binary. ¨ Metadata ¨ The effect of colour depth (number of bits per pixel) and resolution (number of pixels on the image/screen) on: o The quality of the image o The size of an image file	Each pixel has a specific colour, represented by a specific code The effect on image size and quality when changing colour depth and resolution Metadata stores additional image information (e.g. height, width, etc.)
Sound ¨ How sound can be sampled and stored in digital form Sampling is converting analogue audio signals into digital signals. The computer takes measurements of sound wave value at intervals called sampling intervals. The values are converted into digital values to then be saved in a binary.	Analogue sounds must be stored in binary Sample rate – measured in Hertz (Hz) Duration – how many seconds of audio the sound file contains Bit depth – number of bits available to store each sample (e.g. 16-bit)

¨ The effect of sample rate, duration and bit depth on: o The playback quality o The size of a sound file	
1.2.5 Compression The need for compression Types of compression: o Lossy o Lossless	Common scenarios where compression may be needed Advantages and disadvantages of each type of compression Effects on the file for each type of compression Not required Ability to carry out specific compression algorithms

1.3 – Computer networks, connections and protocols	
Sub topic	Guidance and Notes
1.3.1 Networks and topologies ¨ Types of network: o LAN (Local Area Network) o WAN (Wide Area Network) ¨ Factors that affect the performance of networks ¨ The different roles of computers in a client-server and a peer-to-peer network ¨ The hardware needed to connect stand-alone computers into a Local Area Network: o Wireless access points o Routers	Required The characteristics of LANs and WANs including common examples of each Understanding of different factors that can affect the performance of a network, e.g.: Number of devices connected Bandwidth The tasks performed by each piece of hardware The concept of the Internet as a network of computer networks A Domain Name Service (DNS) is made up of multiple Domain Name Servers A DNS's role in the conversion of a URL to an IP address Concept of servers providing services (e.g. Web server " Web pages, File server " file storage/retrieval) Concept of clients requesting/using services from a server The Cloud: remote service provision (e.g. storage, software, processing)

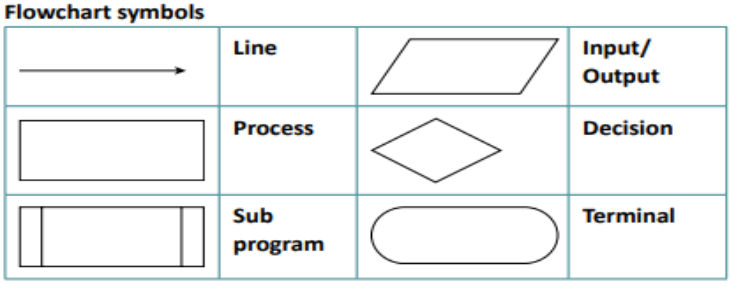
o Switches o NIC (Network Interface Controller/Card) o Transmission media ¨ The Internet as a worldwide collection of computer networks: o DNS (Domain Name Server) o Hosting o The Cloud o Web servers and clients ¨ Star and Mesh network topologies	Advantages and disadvantages of the Cloud Advantages and disadvantages of the Star and Mesh topologies Apply understanding of networks to a given scenario
1.3.2 Wired and wireless networks, protocols and layers Modes of connection: o Wired • Ethernet o Wireless • Wi-Fi • Bluetooth ¨ Encryption ¨ IP addressing and MAC addressing ¨ Standards ¨ Common protocols including: o TCP/IP (Transmission Control Protocol/Internet Protocol) o HTTP (Hyper Text Transfer Protocol) o HTTPS (Hyper Text Transfer Protocol Secure) o FTP (File Transfer Protocol) o POP (Post Office Protocol) o IMAP (Internet Message Access Protocol) o SMTP (Simple Mail Transfer Protocol) ¨ The concept of layers	Compare benefits and drawbacks of wired versus wireless connection Recommend one or more connections for a given scenario The principle of encryption to secure data across network connections IP addressing and the format of an IP address (IPv4 and IPv6) A MAC address is assigned to devices; its use within a network The principle of a standard to provide rules for areas of computing Standards allows hardware/software to interact across different Manufacturers/producers The principle of a (communication) protocol as a set of rules for transferring data That different types of protocols are used for different purposes The basic principles of each protocol i.e. its purpose and key features How layers are used in protocols, and the benefits of using layers; for a teaching example, please refer to the 4-layer TCP/IP model (Not required – details of Ethernet, Wi-Fi and Bluetooth protocols, differences between static and dynamic, or public and private IP addresses, Knowledge of individual standards) Not required but recommended: Knowledge of the names and function of each TCP/IP layer)
1.4 – Network security	
Sub topic	Guidance and Notes
1.4.1 Threats to computer systems and networks Forms of attack:	Threats posed to devices/systems

o Malware o Social engineering, e.g. phishing, people as the 'weak point' o Brute-force attacks o Denial of service attacks o Data interception and theft o The concept of SQL injection	Knowledge/principles of each form of attack including: § How the attack is used § The purpose of the attack
<u>1.4.2 Identifying and preventing vulnerabilities</u> ¨ Common prevention methods: o Penetration testing o Anti-malware software o Firewalls o User access levels o Passwords o Encryption o Physical security	Required Understanding of how to limit the threats posed in 1.4.1 Understanding of methods to remove vulnerabilities Knowledge/principles of each prevention method: § What each prevention method may limit/prevent § How it limits the attack

1.5 – Systems software	
Sub topic	Guidance and Notes
1.5.1 Operating systems ¨ The purpose and functionality of operating systems: o User interface o Memory management and multitasking o Peripheral management and drivers o User management o File management	What each function of an operating system does Features of a user interface Memory management, e.g. the transfer of data between memory, and how this allows for multitasking Understand that: § Data is transferred between devices and the processor § This process needs to be managed User management functions, e.g.: § Allocation of an account § Access rights § Security, etc. File management, and the key features, e.g.: § Naming § Allocating to folders § Moving files § Saving, etc. (Not required - Understanding of paging or segmentation)
1.5.2 Utility software The purpose and functionality of utility software Utility system software: o Encryption software o Defragmentation o Data compression	Understand that computers often come with utility software, and how this performs housekeeping tasks Purpose of the identified utility software and why it is required

1.6 – Ethical, legal, cultural and environmental impacts of digital technology	
Sub topic	Guidance and Notes
<u>1.6.1 Ethical, legal, cultural and environmental impact</u> Impacts of digital technology on wider society including: o Ethical issues o Legal issues o Cultural issues o Environmental issues o Privacy issues ¨ Legislation relevant to Computer Science: o The Data Protection Act 2018 o Computer Misuse Act 1990 o Copyright Designs and Patents Act 1988 o Software licences (i.e. open source and proprietary)	Technology introduces ethical, legal, cultural, environmental and privacy issues Knowledge of a variety of examples of digital technology and how this impacts on society An ability to discuss the impact of technology based around the issues listed The purpose of each piece of legislation and the specific actions it allows or prohibits The need to license software and the purpose of a software licence Features of open source (providing access to the source code and the ability to change the software) Features of proprietary (no access to the source code, purchased commonly as off-the-shelf) Recommend a type of licence for a given scenario including benefits and drawbacks

2c. Content of Computational thinking, algorithms and programming (J277/02)

2.1 – Algorithms	
Sub topic	Guidance and Notes
2.1.1 Computational thinking ¨ Principles of computational thinking: o Abstraction o Decomposition o Algorithmic thinking	Understanding of these principles and how they are used to define and refine problems When solving a problem (any problem) these principles can be useful: **Decomposition** is the process of analyzing a problem or solution into logical parts so that solutions to these different modules can be created and tested in stages and maybe by a team of people. **Abstraction** is the naming and separating of the parts of a process/system/solution so that the problem can be solved one module at a time which usually is easier as each part is less complex. OCR seem to prefer this definition. "Hiding or removing irrelevant details from a problem to reduce complexity." **Algorithmic thinking** is used to work out the processes needed perform a particular function or module.
2.1.2 Designing, creating and refining algorithms ¨ Identify the inputs, processes, and outputs for a problem ¨ Structure diagrams ¨ Create, interpret, correct, complete, and refine algorithms using: o Pseudocode o Flowcharts o Reference language/high-level programming language ¨ Identify common errors ¨ Trace tables	Produce simple diagrams to show: § The structure of a problem § Subsections and their links to other subsections https://www.youtube.com/watch?v=F6f6W7S9Y6k Structure diagrams can be used to illustrate the decomposition of a problem/solution into modules. The structure diagram is like an upside down tree. The whole problem at the top then subdivided into its constituent parts. Complete, write or refine an algorithm using the techniques listed Identify syntax/logic errors in code and suggest fixes Create and use trace tables to follow an algorithm **Flowchart symbols** 

2.1.3 Searching and sorting algorithms ¨ Standard searching algorithms: o Binary search o Linear search ¨ Standard sorting algorithms: o Bubble sort o Merge sort o Insertion sort	Understand the main steps of each algorithm Understand any pre-requisites of an algorithm Apply the algorithm to a data set Identify an algorithm if given the code or pseudocode for it https://www.cs.usfca.edu/~galles/visualization/Search.html (Not required - the Exam Reference Language algorithm for Merge Sort) https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

2.2 – Programming fundamentals

<table>
<tr><th>Sub topic</th><th>Guidance and Notes</th></tr>
<tr><td>2.2.1 Programming fundamentals
¨ The use of variables, constants, operators, inputs, outputs and assignments
¨ The use of the three basic programming constructs used to control the flow of a program:
o Sequence
o Selection
o Iteration (count- and condition-controlled loops)
¨ The common arithmetic operators
¨ The common Boolean operators AND, OR and NOT</td><td>Practical use of the techniques in a high-level language within the classroom
Understanding of each technique
Recognise and use the following operators:
<table>
<tr><th colspan="2">Comparison operators</th><th colspan="2">Arithmetic operators</th></tr>
<tr><td>==</td><td>Equal</td><td>+</td><td>Addition</td></tr>
<tr><td>!=</td><td>Not equal</td><td>–</td><td>Subtraction</td></tr>
<tr><td><</td><td>Less than</td><td>*</td><td>Multiplication</td></tr>
<tr><td><=</td><td>Less than or equal to</td><td>/</td><td>Division</td></tr>
<tr><td>></td><td>Greater than</td><td>MOD</td><td>Modulus</td></tr>
<tr><td>>=</td><td>Greater than or equal to</td><td>DIV</td><td>Quotient (Integer division)</td></tr>
<tr><td></td><td></td><td>^</td><td>Exponentiation (to the power)</td></tr>
</table></td></tr>
<tr><td>2.2.2 Data types
¨ The use of data types:
o Integer
o Real
o Boolean
o Character and string
o Casting</td><td>Practical use of the data types in a high-level language within the classroom.
Ability to choose suitable data types for data in a given scenario.
Understand that data types may be temporarily changed through casting, and where this may be useful.</td></tr>
<tr><td>2.2.3 Additional programming techniques
¨ The use of basic string manipulation

¨ The use of basic file handling operations:
o Open
o Read
o Write</td><td>Practical use of the additional programming techniques in a high-level language within the classroom.

Ability to manipulate strings, including:
§ Concatenation
§ Slicing</td></tr>
</table>

o Close ¨ The use of records to store data ¨ The use of SQL to search for data ¨ The use of arrays (or equivalent) when solving problems, including both one-dimensional (1D) and two-dimensional arrays (2D) ¨ How to use sub programs (functions and procedures) to produce structured code ¨ Random number generation	Records can be used to store set of related data. Structured Query Language is a Language that allows you to manipulate data in database tables. Arrays as fixed length or static structures. Use of 2D arrays to emulate database tables of a collection of fields, and records. The use of functions and procedures Where to use functions and procedures effectively The use of the following within functions and procedures: § local variables/constants § global variables/constants § arrays (passing and returning) SQL commands SELECT, FROM, WHERE: SELECT FIELD1, FIELD2 (or *) FROM TABLE WHERE CONDITION CONDITION could be written FIELD < ARGUMENT for example RESULT = 10 or SCORE > 50 Be able to create and use random numbers in a program

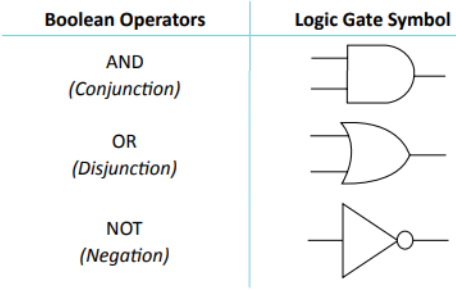
2.3 – Producing robust programs

Sub topic	Guidance and Notes
2.3.1 Defensive design ¨ Defensive design considerations: o Anticipating misuse o Authentication ¨ Input validation ¨ Maintainability: o Use of sub programs o Naming conventions o Indentation o Commenting	Understanding of the issues a programmer should consider to ensure that a program caters for all likely input values Understanding of how to deal with invalid data in a program Authentication to confirm the identity of a user Practical experience of designing input validation and simple authentication (e.g. username and password) Understand why commenting is useful and apply this appropriately

2.3.2 Testing ¨ The purpose of testing ¨ Types of testing: o Iterative o Final/terminal ¨ Identify syntax and logic errors ¨ Selecting and using suitable test data: o Normal o Boundary o Invalid/Erroneous ¨ Refining algorithms	The difference between testing modules of a program during development and testing the program at the end of production Syntax errors as errors which break the grammatical rules of the programming language and stop it from being run/translated Logic errors as errors which produce unexpected output **Normal** test data as data which should be accepted by a program without causing errors **Boundary** test data as data of the correct type which is on the very edge of being valid. **Invalid** test data as data of the correct data type which should be rejected by a computer system **Erroneous** test data as data of the incorrect data type which should be rejected by a computer system Ability to identify suitable test data for a given scenario Ability to create/complete a test plan

2.4 – Boolean logic

Sub topic	Guidance
2.4.1 Boolean logic ¨ Simple logic diagrams using the operators AND, OR and NOT ¨ Truth tables ¨ Combining Boolean operators using AND, OR and NOT ¨ Applying logical operators in truth tables to solve problems	Knowledge of the truth tables for each logic gate Recognition of each gate symbol Understanding of how to create, complete or edit logic diagrams and truth tables for given scenarios Ability to work with more than one gate in a logic diagram

	 **Truth Tables** AND: A | B | A AND B — 0 0 0; 0 1 0; 1 0 0; 1 1 1 OR: A | B | A OR B — 0 0 0; 0 1 1; 1 0 1; 1 1 1 NOT: A | NOT A — 0 1; 1 0 Alternatives Use of other valid notation will be accepted within the examination, e.g. Using T/F for 1/0, or V for OR, etc.

2.5 – Programming languages and Integrated Development Environments	
Sub topic	Guidance
2.5.1 Languages ¨ Characteristics and purpose of different levels of programming language: o High-level languages o Low-level languages ¨ The purpose of translators ¨ The characteristics of a compiler and an interpreter	The differences between high- and low-level programming languages The need for translators The differences, benefits and drawbacks of using a compiler or an interpreter (Not required - Understanding of assemblers)
2.5.2 The Integrated Development Environment (IDE) ¨ Common tools and facilities available in an Integrated Development Environment (IDE): o Editors o Error diagnostics o Run-time environment o Translators	Knowledge of the tools that an IDE provides How each of the tools and facilities listed can be used to help a programmer develop a program Practical experience of using a range of these tools within at least one IDE

2d. Practical Programming skills

- Design
- Write (Code)
- Test
- Refine

Any high-level text-based programming language, such as:

- Python
- C family of languages (C#, C++, etc.)
- Java
- JavaScript
- Visual Basic/.Net
- PHP
- Delphi
- BASIC

Students should have experience of all the Practical Programming skills so schools are encouraged to consider using a second language for practical experience.

Practical Programming skills will be assessed in Component 2 of the qualification.

Question focus	Questions asked in:	Students respond using:
Design	Natural English	✓ Pseudocode ✓ Flowcharts ✓ Tick-box responses ✓ Natural English
Write	Pseudocode Natural English Flowcharts	✓ OCR Exam Reference Language ✓ A high-level programming language
Test	OCR Exam Reference Language	✓ Trace tables ✓ Creating test plans ✓ Identifying suitable test data
Refine	OCR Exam Reference Language	✓ OCR Exam Reference Language ✓ A high-level programming language ✓ Natural English

OCR Exam Reference Language

Operators

Comparison operators

==	Equal to	<=	Less than or equal to
!=	Not equal to	>	Greater than
<	Less than	>=	Greater than or equal to

Arithmetic operators

+	Addition	/	Division
-	Subtraction	MOD	Modulus
*	Multiplication	DIV	Quotient
^	Exponent		

Boolean operators

AND	Logical AND
OR	Logical OR
NOT	Logical NOT

Concept	Keyword(s)/Symbols	Example
Commenting		

Comment	//	//This function squares a number
		function squared(number)
		  squared = number^2
		  return squared
		endfunction
		//End of function

Variables		
Assignment	=	x = 3 name = "Louise"
Constants	const	const vat = 0.2
Global Variables	global	global userID = "Cust001"
Input/Output		
Input	input(…)	myName = input("Please enter a name")
Output	print(…)	print("My name is Noni") print(myArray[2,3])
Casting		
Converting to another data type	str()	str(345)
	int()	int("3")
	float()	float("4.52")
	real()	real("4.52")
	bool()	bool("True")

Concept	Keyword(s)/Symbols	Example
Iteration		

FOR loop
(Count-controlled)
	`for … to …`
	
	`next …`
	
	`for … to … step …`
	
	`next …`
		`for i=0 to 9`
		`  print("Loop")`
		`next i`
		This will print the word "Loop" 10 times, i.e. 0-9 **inclusive**.
		
		`for i=2 to 10 step 2`
		`  print(i)`
		`next i`
		This will print the even numbers from 2 to 10 inclusive.
		
		`for i=10 to 0 step -1`
		`  print(i)`
		`next i`
		This will print the numbers from 10 to 0 inclusive, i.e. 10, 9, 8,…, 2, 1, 0.
		
		Note that the `'step'` command can be used to increment or decrement the loop by any positive or negative integer value.

WHILE loop
(Condition-controlled)
	`while …`
	
	`endwhile`
		`while answer != "Correct"`
		`  answer = input("New answer")`
		`endwhile`
		Will loop until the user inputs the string "Correct". Check condition is carried out before entering loop.

DO WHILE loop
(Condition-controlled)
	`do`
	
	`until …`
		`do`
		`  answer = input("New answer")`
		`until answer == "Correct"`
		Will loop until the user inputs the string "Correct". Loop iterates once before a check is carried out.

Concept	Keyword(s)/Symbols	Example
Selection		

IF-THEN-ELSE	`if … then`
	`elseif … then`
	`else`
	`endif`
		`if answer == "Yes" then`
		`  print("Correct")`
		`elseif answer == "No" then`
		`  print("Wrong")`
		`else`
		`  print("Error")`
		`endif`

CASE SELECT or
SWITCH
	`switch … :`
	`  case … :`
	`  case … :`
	`  default:`
	`endswitch`
		`switch day :`
		`  case "Sat":`
		`  print("Saturday")`
		`  case "Sun":`
		`  print("Sunday")`
		`  default:`
		`  print("Weekday")`
		`endswitch`

Concept	Keyword(s)/Symbols	Example
String handling/operations		
String length	`.length`	`subject = "ComputerScience"` `subject.length` **gives** the value 15

Substrings	`.substring(x , i)`
	`.left(i)`
	`.right(i)`
		`subject.substring(3,5)` **returns** `"puter"`
		`subject.left(4)` **returns** `"Comp"`
		`subject.right(3)` **returns** `"nce"`
		x is starting index; i is number of characters; 0 indexed

Concatenation	`+`	`print(stringA + stringB)` `print("Hello, your name is: " + name)`
Uppercase	`.upper`	`subject.upper` **gives** `"COMPUTERSCIENCE"`
Lowercase	`.lower`	`subject.lower` **gives** `"computerscience"`
ASCII Conversion	`ASC(…)` `CHR(…)`	`ASC(A)` **returns** `65` (numerical) `CHR(97)` **returns** `'a'` (char)

Concept	Keyword(s)/Symbols	Example
File handling		
Open	`open(…)`	`myFile = open("sample.txt")` **Note that the file needs to be stored as a variable.**
Close	`.close()`	`myFile.close()`
Read line	`.readLine()`	`myFile.readLine()` **returns** the next line in the file
Write line	`.writeLine(…)`	`myFile.writeLine("Add new line")` **Note that the line will be written to the END of the file.**

End of file	`.endOfFile()`	`while NOT myFile.endOfFile()`
		`  print(myFile.readLine())`
		`endwhile`

Create a new file	`newFile()`	`newFile("myText.txt")` **Creates a new text file called "myText". The file would then need to be opened using the above command for Open.**
Arrays		

Declaration	`array colours[…]`	`array colours[5]`
		Creates 1D array with 5 elements (index 0 to 4).
		`array colours = ["Blue", "Pink", "Green", "Yellow", "Red"]`
		Arrays can be declared with values assigned.

Arrays are **0 indexed** Arrays only store a **single data type**	`array gameboard[…,…] = …`	`array gameboard[8,8]` **Creates 2D array with 8 elements (index 0 to 7).**
Assignment	`names[…] = …` `gameboard[…,…] = …`	`names[3] = "Noni"` `gameboard[1,0] = "Pawn"`

Concept	Keyword(s)/Symbols	Example
Sub programs		

Procedure	`procedure name(…)`
	
	`endprocedure`
		`procedure agePass()`
		`  print("You are old enough to ride")`
		`endprocedure`
		
		`procedure printName(name)`
		`  print(name)`
		`endprocedure`
		
		`procedure multiply(num1, num2)`
		`  print(num1 * num2)`
		`endprocedure`

Calling a procedure	`procedure(parameters)`	`agePass()`
		
		`printName(parameter)`
		
		`multiply(parameter1, parameter2)`

Function	`function name(…)`
	`  …`
	`  return …`
	`endfunction`
		`function squared(number)`
		`  squared = number^2`
		`  return squared`
		`endfunction`

Calling a function	`function(parameters)`	`print(squared(4))`
		
		`newValue = squared(4)`
		Note: Function returns should be stored in a variable if needed for later use in a program.

Random numbers		

Random numbers	`random(…,…)`	`myVariable = random(1,6)`
		Creates a random integer between 1 and 6 inclusive.
		
		`myVariable = random(-1.0,10.0)`
		Creates a random real number between -1.0 and 10.0 inclusive.