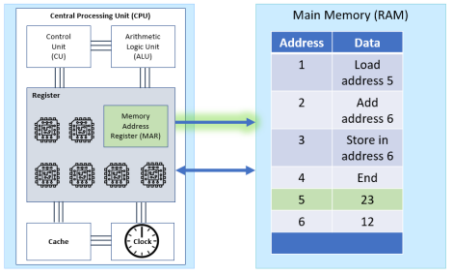
GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of variables, constants, inputs, outputs and assignments

Key Terminology	BCS Definition
Variable	"A value that can change depending on conditions or information passed to the program."
Constant	"A value that cannot be altered by the program during normal execution."
Operator	"Tells a program how to manipulate or interpret values. Categories of operators you need to know about are arithmetic, Boolean and comparison."
Assignment	"Giving a variable or constant a value (e.g., counter = 0)."

What is a variable?

- The memory address register (MAR) contains the address of an instruction or data to be fetched from or written to main memory.
- In a similar way, a variable is nothing more than a pointer to a memory address with a user-friendly label.



Key Terms

- **Variable**: A value stored in memory that can change while the program is running – can be an integer, character, string, real/float or Boolean value.
- **Constant**: A value that is assigned when the program is first written and does not change while it is running.
- **Assignment**: Supplying a variable or constant with a value.
- **Casting**: Converting a variable from one data type to another (e.g., integer to string). A variable can be an integer, character, string, real (float) or Boolean.
- **Input**: A value read from an input device (e.g., keyboard).
- **Output**: Data generated by the computer and displayed to the user.

The advantages of constants

- Constants make a program easier to read, as they are usually declared and assigned at the top of the program.
- They allow programmers to modify a program by changing one value rather than having to change every instance of a value throughout the program, reducing the possibility of errors.
- If constants are used instead of variables, a compiler can be used to optimise code, making the program run faster.

Why do we use casting?

- Casting changes a variable from one data type another (e.g., a string to an integer).
- Inputs from the keyboard are always characters – multiple characters are called a string.
- However, to perform an addition, the arithmetic logic unit (ALU) must use numbers – therefore, a string needs to be *cast* to a number.
- The character "1" as typed on a keyboard is stored as the binary number 00110001 using the ASCII character set.
- Meanwhile, the number 1 that the ALU needs to perform a calculation is stored as the binary number 00000001.
- Integers require less memory than numbers with a decimal part (real numbers), so it makes sense to use integers where we can to make a program more memory-efficient.
- However, it may be necessary to cast an integer to a real number in a program – some commands also require data to be of a particular data type.

Why do we use casting?

Integer
A positive or negative whole number (e.g., 6)
Use: total = total + score
Never used for telephone numbers
Used only when arithmetic needs to be performed on the data

Real
A number with a decimal part (e.g., 6.5)
Sometimes called float (floating-point)
Use: cost = total + vat

Character
A single alphanumeric character (e.g., "1" or "a")
Used for menu choices
Use: choice = input("Enter your choice:")

String
Multiple alphanumeric characters (e.g., "Craig")
Used for words and telephone numbers
Use: forename = "Craig"

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of the three basic programming constructs

Key Terminology	BCS Definition
Sequence	"One of the three basic programming constructs. Instructions that are carried one after the other in order."
Selection	"One of the three basic programming constructs. Instructions that can evaluate a Boolean expression and branch off to one or more alternative paths."
Count-controlled iteration	"An iteration that loops a fixed number of times. A count is kept in a variable called an index or counter. When the index reaches a certain value (the loop bound) the loop will end. Count-controlled repetition is often called definite repetition because the number of repetitions is known before the loop begins executing."
Condition-controlled iteration	"A way for computer programs to repeat one or more steps depending on conditions set either a) initially by the programmer or b) by the program during execution."

1. Sequence

- Sequence means executing instructions in order, one after the other.

```
#Import libraries
import random

#Constants
rolls_per_player = 2

#Initialise varilables
total = 0

#Output the dice
print("dice 1:",dice1)
print("dice 2:",dice2)

#Cast the dice values from integers to strings ready for changing
the order of the dice
dice1_string = str(dice1)
dice2_string = str(dice2)
```

2. Selection (branching)

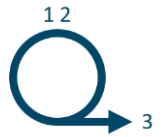
- Selection means a program will branch depending on certain conditions.


```
#Change the order of the dice to the highest value first and join
the values together
if dice1 > dice2:
   roll_value = dice1_string + dice2_string
else:
   roll_value = dice2_string + dice1_string



#Output the value of the dice
if user input == roll_value:
   print("You worked it out correctly.")
else:
   print("No, the value of the roll is:",roll_value)
```

3. Iteration (looping)



- Iteration, sometimes called looping, means repeating sections of code.

A **FOR loop** (also known as a **counter-controlled loop**) is used when the required number of iterations is known ahead of execution.

```
#Roll the dice
for rolls in range(rolls_per_player):
   dice1 = 0
   dice2 = 0
   #Handle the possibility of a double throw
   while dice1 == dice2:
      dice1 = random.randint(1,6)
      dice2 = random.randint(1,6)
```

A **WHILE loop** (also known as a **condition-controlled loop**) is used when the required number of iterations is not known ahead of execution because the variable used to determine when the iteration ends is changing within the iteration itself.

```
WHILE answer != "computer"
   answer ← USERINPUT("What is the password?")
ENDWHILE
```

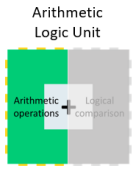
A **DO...UNTIL loop** is an alternative to WHILE where the code executes at least once before the condition is checked.

```
DO
   answer ← USERINPUT("What is the password?")
UNTIL answer = "computer"
```

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The common arithmetic, comparison and Boolean operators

Key Terminology	BCS Definition
Arithmetic operator	+ - / * ^ "Used in mathematical expressions (e.g., num1 + num2 = sum)."
Boolean operator: AND	"A logical operator used within a program. Only returns TRUE if both values being compared are TRUE."
Boolean operator: OR	"A logical operator used within a program. Returns TRUE as long as either value being compared is TRUE."
Boolean operator: NOT	"A way for computer programs to repeat one or more steps depending on conditions set either a) initially by the programmer or b) by the program during execution."
Arithmetic operator: MOD	"Integer division. MOD outputs the remainder left over after division – e.g., 10 MOD 3 = 1."
Arithmetic operator: DIV	"Integer division: DIV outputs the number of times a number fits into another number – e.g., 10 DIV 3 = 3."

1. Common arithmetic operators



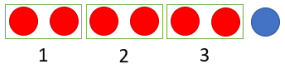
Operator	Example	Meaning	Result
+	x = 7 + 2	Addition	9
-	x = 7 - 2	Subtraction	5
*	x = 7 * 2	Multiplication	14
/	x = 7 /2	Division	3.5
^	x = 7 ^ 2	Exponentiation	49
MOD	x = 7 MOD 2	Modulus	1
DIV	x = 7 DIV 2	Integer division	3

7 MOD 2

What is the remainder once you fit the number on the right into the number on the left as many times as possible?

7 DIV 2

How many times does the number on the right fit into the number on the left?

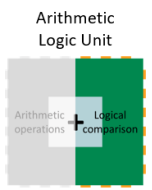


2. Common comparison operators

Operator	Example	Meaning
==	7 == 7	Equal to
!=	7 != 5	Not equal to
<	5 < 7	Less than
<=	5 <= 5	Less than or equal to
>	7 > 5	Greater than
>=	7 >= 7	Greater than or equal to

Note: Different languages may use alternate symbols to represent these operators. For example, "not equal to" could be:

!=
<>
~=



3. Boolean operators

Boolean operators can be used in If statements and while loops to check if a condition is true or false. – See SLR 2.4 Boolean Logic

NOT

end_of_file = False
while **not** end_of_file ...is the same as... while end_of_file == **False:**

AND

end_of_file = False
found = False
while **not** end_of_file **and not** found

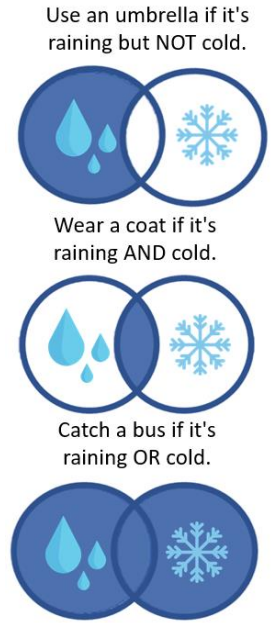
OR

```
if x == 2 or x == 4

while choice < 1 or choice > 3
```

Creating an infinite loop

```
while True
```



GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of data types and casting

Key Terminology	BCS Definition
Data type	"The basic data types provided as building blocks by a programming language. Most languages allow for more complicated, composite types to be constructed from basic types recursively – e.g., char, integer, float, Boolean. As an extension, a string data type is constructed behind the scenes of many char data types."
Integer	"A data type used to store positive and negative whole numbers."
Real	"A data type used to store an approximation of a real number in a way that can support a trade-off between range and precision. Typically, a number is represented approximately to a fixed number of significant digits and scaled using an exponent."
Boolean	"Used to store logical conditions – e.g., TRUE/FALSE, ON/OFF, YES/NO, etc."
Character	"A single alphanumeric symbol."
String	"A sequence of alphanumeric characters and/or symbols – e.g., a word or sentence."
Casting	"Converting a variable from one data type to another. For example, a variable entered as a string needs to be an integer for calculation – age = INPUT("Enter your age: ") age = INT(age)."

Variables and constants can hold:

Integer: A whole positive or negative number. Only used for data that requires calculations.
Real/float: A number with a decimal part. Only used for data that requires calculations.
Character: A single alphanumeric character.
String: A set of characters. Used for all data that is not calculated.
Boolean: True or false, often called a flag. Used to track if something has happened or not.

Changing a data type from one type to another is known as casting, which allows numbers to be manipulated as strings and is used to ensure that sub-problems receive data in a format they are expecting.

Casting also allows inputs that are always strings to become numbers:
x = str(x) casts the variable x to a string
x = int(x) casts the variable x to an integer
x = float(x) cast the variable x to a float
x = chr(x) cast the variable x to a character

```
def roll_a_dice():
   #Roll a dice
   return random.randint(1,6)

def order_dice(d1, d2):
   #Change the order of the dice to highest value first
   #and join the values together
   if d1 >= d2:
      return int(d1 + d2)
   else:
      return int(d2 + d1)

def output_throw():
   #Output the results of the two dice thrown
   dice1 = 0
   dice2 = 0
   while dice1 == dice2:
      dice1 = str(roll_a_dice())
      dice2 = str(roll_a_dice())
   print("Dice 1:", dice1)
   print("Dice 2:", dice2)
   roll = order_dice(dice1, dice2)
   return roll

#Main program
#This tool outputs the value of two dice
import random
print(output_throw())
```

Selecting suitable data types for data in a given scenario

Select the most appropriate data types for the variables in this scenario:

"A program asks for a user's name, age and telephone number. If they are 17 years of age or younger, the program informs them they are not eligible to vote as they are too young. If they are 18 or over, the program stores their details and tells them they are eligible to vote."

	String	Integer	Real	Boolean
vote				✓
name	✓			
age		✓		
telephoneNumber	✓			

Many students often choose integer by mistake.

The telephone number 07548 433844 would be stored as 7548433844 if we used an integer, so we would lose the leading 0.

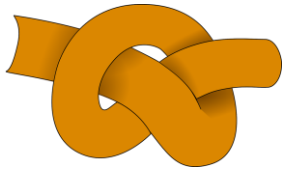
Numbers are also often entered with other non-integer symbols – e.g., (07548) 433-844.

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of basic string manipulation: OCR Exam Reference Language (ERL)

Key Terminology	BCS Definition
String manipulation	"Commands and techniques that allow you to alter and extract information from textual strings – e.g., .length .substring(x, i) .left(i) .right(i) .upper .lower ASC(…) CHR(…)."

What is String Manipulation?

String manipulation is the act of manipulating, extracting or changing the characters in a string variable.



To get the length of a string:

Concept	Keyword/Symbol
String length	**.length**

Returns the length of a string.

```
length = string.length
```

e.g., if the string was "Hello", the length would be 5.

Converting cases:

Concept	Keyword/Symbol
Uppercase	**.upper**
Lowercase	.lower

Returns the string in uppercase or lowercase.

```
ustring = string.upper
lstring = string.lower
```

e.g., if the string was "Hello", ustring would be "HELLO".
e.g., if the string was "Hello", lstring would be "hello".

To extract a substring:

Concept	Keyword/Symbol
Substrings	**.substring(x, i)**

Returns part of a string, starting at the character of the first parameter and counting up by the number in the second parameter.

```
stringname.subString(startingPosition, numberOfCharacters)
chars = string.substring(3,1)
```

e.g., if the string was "Hello", chars would be "l".

To extract text from the left or right of a string:

Concept	Keyword/Symbol
Substrings	**.left(i)**
Substrings	**.right(i)**

Returns the left most or right most characters from a string where the parameter indicates how many to return.

```
left = string.left(8)
```

e.g., if the string was "ComputerScience", then .left(8) would return "Computer".

```
right = string.right(7)
```

e.g., if the string was "ComputerScience", then .right(7) would return "Science".

To concatenate (join) separate strings together:

Concept	Keyword/Symbol
Concatenation	**+**

Joins separate string values together

```
Print(stringA + stringB)
```

```
Print("Hello, your name is: " + name)
```

ASCII conversion:

Concept	Keyword/Symbol
ASCII Converstion	**ASC(…)**
	CHR(…)

Returns the ASCII value of a character.
Returns a character from its ASCII number.

```
ASC(character)
ascii = ASC("A")
CHR(asciinumber)
char = CHR(65)
```

e.g., ascii would be 65.

e.g., char would be "A".

Example:

```
someText="Computer Science"
print(someText.length)
print((someText.substring(3,3)).upper)
```

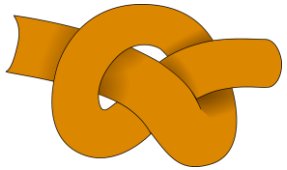
16
PUT

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of basic string manipulation: Python

Key Terminology	BCS Definition
String manipulation	"Commands and techniques that allow you to alter and extract information from textual strings – e.g., .length .substring(x, i) .left(i) .right(i) .upper .lower ASC(…) CHR(…)."

What is String Manipulation?

String manipulation is the act of manipulating, extracting or changing the characters in a string variable.



To get the length of a string:

Concept	Keyword/Symbol
String length	**Len(string)**

Returns the length of a string.

```
length = len(string)
```

e.g., if the string was "Hello", the length would be 5.

Converting cases:

Concept	Keyword/Symbol
Uppercase	**.upper()**
Lowercase	.lower()

Returns the string in uppercase or lowercase.

```
ustring = string.upper()
lstring = string.lower()
```

e.g., if the string was "Hello", ustring would be "HELLO".
e.g., if the string was "Hello", lstring would be "hello".

To extract a substring:

Concept	Keyword/Symbol
Substrings	**.substring[x, i]**

Returns part of a string, starting at the character of the first parameter and counting up by the number in the second parameter.

```
stringname.subString(startingPosition, numberOfCharacters)
chars = string[2:3]
```

e.g., if the string was "Hello", chars would be "l".

To extract text from the left or right of a string:

Concept	Keyword/Symbol
Substrings	**.left(i)**
Substrings	**.right(i)**

Returns the left most or right most characters from a string where the parameter indicates how many to return.

```
left = string.left(8)
```

e.g., if the string was "ComputerScience", then .left(8) would return "Computer".

```
right = string.right(7)
```

e.g., if the string was "ComputerScience", then .right(7) would return "Science".

To concatenate (join) separate strings together:

Concept	Keyword/Symbol
Concatenation	**+**

Joins separate string values together

```
print(stringA + stringB)
```

```
print("Hello, your name is: " + name)
```

ASCII conversion:

Concept	Keyword/Symbol
ASCII Converstion	**ord(…)**
	chr(…)

Returns the ASCII value of a character.
Returns a character from its ASCII number.

```
ord(character)
ascii = ord("A")
chr(asciinumber)
char = chr(65)
```

e.g., ascii would be 65.

e.g., char would be "A".

Example:

```
someText="Computer Science"
print(len(someText))
print((someText[3:6].upper())
```

16
PUT

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of basic file handling operations

Key Terminology	BCS Definition
File handling: Open	"File handling is the process of dealing with input to and from files. Files first have to be opened, creating a handle to the file and allowing reading and writing."
File handling: Read	"Once a file has been opened, it is possible to use commands to read its contents and return them to a program."
File handling: Write	"Once a file has be opened it is possible to use commands to write data to the file from a program."
File handling: Close	"When a file is no longer in use, closing it releases the file handle and breaks the connection between the file and a program."

Reading and writing data to text files

The stages of writing data to a file are:

- Open the file for creating/overwriting or appending to a file.
- Write the data to the file.
- Close the file.

The stages of reading data from a file are:

- Open the file for reading data.
- Set a Boolean variable to "false" to indicate the end of file has not been reached.
- While the end of file flag is false and the search item has not been found:
    - Read the data from the file.
    - If the data matches what is being searched for, assign the data to variables or output.
    - Check if the end of the file has been reached and if it has, set the Boolean variable to "true".
- Close the file.

Writing data to text files

```
name = input("Enter the name of the character: ")
health = input("Enter the health of the character: ")
stamina = input("Enter the stamina of the character: ")
hunger = input("Enter the hunger of the character: ")

#Step 1 - Open the file for characters
f = open("characters.txt", "a")

#Step 2 - Write the character data to the file
f.write(name+"\n")
f.write(health+"\n")
f.write(stamina+"\n")
f.write(hunger+"\n")

#Step 3 - Close the file of characters
f.close()
```

Reading data from text files

```
#Input the character to find in the file
character = input("Which character do you wish to output? ")

#Step 1 - Open the file for reading/writing
f = open("characters.txt", "r")

#Step 2 - Use a flag to indicate if the end of file is reached
end_of_file = False

#Step 3 - Iterate through all the data in the file
while not end_of_file:
    name = f.readline().strip()
    health = f.readline().strip()
    stamina = f.readline().strip()
    hunger = f.readline().strip()

    #Step 4 - End of file or output data from file
    if character == name:
        print(name)
        print("Health:",health)
        print("Stamina:",stamina)
        print("Hunger:",hunger)
    if name == "":
        end_of_file = True

#Step 5 - Close the file
f.close()
```

Reading and writing data to text files (OCR Exam Reference Language)

We use "open" to open a file to read/write data from/to. We then use writeLine to write a line of text and readLine to return a line of text. The following program assigns x as the first line of sample.txt.

```
myFile = open("sample.txt")
   x = myFile.readLine()
myFile.close()
```

endOfFile() is used to determine the end of the file. The following program will print out the contents of sample.txt.

```
myFile = open("sample.txt")
while NOT myFile.endOfFile()
   print(myFile.readLine())
endwhile
myFile.close()
```

In the program below, hello world is written to sample.txt – any previous content is overwritten.

```
myFile = open("sample.txt")
myFile.writeLine("Hello World")
myFile.close()
```

To create a new file called myNewFile.txt.

```
newFile = ("myNewFile.txt")
```

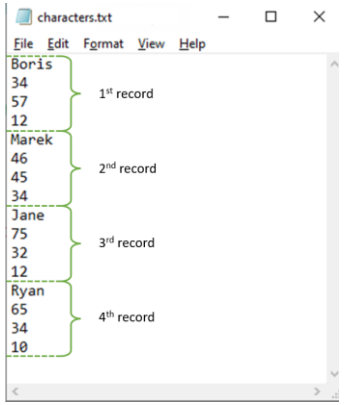
Concept	Keyword/Symbol
Open	open(…)
Close	.close()
Read line	.readLine()
Write line	.writeLine(…)
End of file	.endOfFile()
Create a new file	newFile()

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
The use of records to store data

Key Terminology	BCS Definition
Record	"A data structure consisting of a collection of elements, typically in fixed number and sequence and indexed by name. Elements of records may be called fields. The record is a data type that describes such values and variables. Most modern languages allow programmers to define new record types, as well as specifying the data type of each field and an identifier by which it can be accessed."

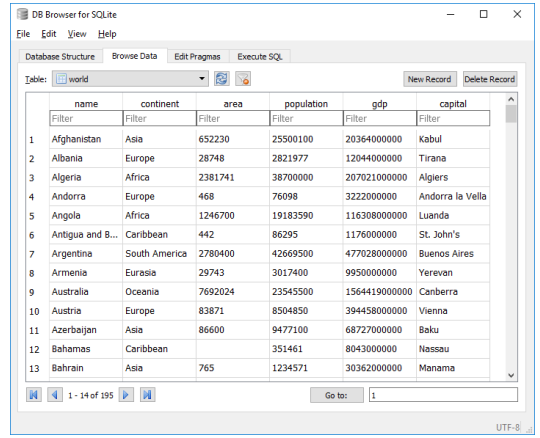
Records stored in text files

- Stored on secondary storage (hard disk/SSD/flash).
- Used to store data when the application is closed.
- Useful for small volumes of data – e.g., configuration files.
- Each entry is stored on a new line or separated with an identifier such as a comma or tab.
- May require a linear search to find/read data, which can be slow with unordered data or record structures.
- Structured text files (e.g., CSV, XML, JSON) can be used to store data and exchange it between applications.



Records stored in databases

- Often stored on remote servers.
- Often used to store data shared by multiple users – e.g., ticket booking system.
- Data is stored in records and fields.
- Use advanced data structures to store data efficiently.
- Data can be searched and sorted using highly efficient algorithms.
- More secure than text files.
- The order of database fields is independent of the code.



Records stored in arrays and lists

- Stored in main memory (RAM).
- Used to store data when a program is running.
- Useful for small volumes of data currently in use by an algorithm.
- Can be single- or multi-dimensional.
- Use indexes to refer to data items.
- Efficient algorithms or linear searches can be used to find data.

countries		
Index	0	1
0	Angola	1246700
1	Austria	83871
2	Belgium	30528

Record structure

- A collection of related fields.
- A field is a variable.
- Each field in a record can have a different data type.
- There are three steps to using record structures:
- Define the record structure – what fields will be in the record?
- Declare a variable or array to use with the record structure.
- Assign and retrieve data from the variable record.
- Be careful to note the dot syntax when using records – e.g., Record<dot>Field, car1.Make

TCar					
Reg_plate	Make	Model	Price	Engine_size	Petrol
VK134KE	Fiat	Punto	14000	1.2	True

Step 1 — **Define** the record structure – what fields will be in the record?

Step 2 — **Declare** a variable or array to use with the record structure

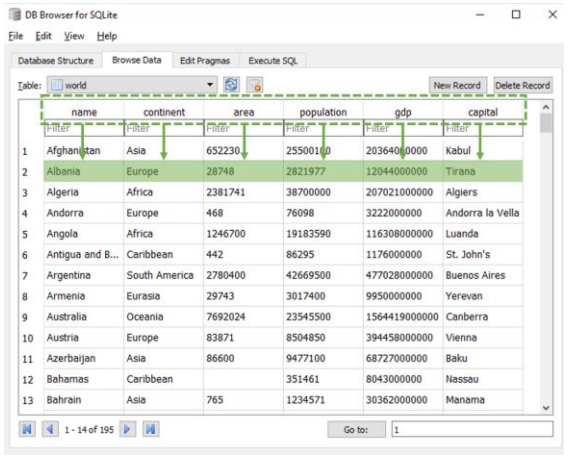
Step 3 — **Assign** and retrieve data from the variable record

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
Using SQL to search for data

Key Terminology	BCS Definition
SQL	"The language and syntax used to write and run database queries."
SQL command: SELECT	"A SQL keyword used query (retrieve) data." **SELECT** Name, Age, Class FROM Students_table WHERE Gender = "Male"
SQL command: FROM	"A SQL keyword used to signify which table(s) are included in a query." SELECT Name, Age, Class **FROM** Students_table WHERE Gender = "Male"
SQL command: WHERE	"A SQL keyword used to filter query results." SELECT Name, Age, Class FROM Students_table **WHERE** Gender = "Male"

What is SQL: Structured Query Language

- Information in a database is stored in records. Each record can contain a number of fields.
- We can manipulate the data in these fields using a programming language called Structured Query Language (SQL).



SQL: Structured Query Language

- SQL is used to create, delete, modify and manipulate records in a database. Basic commands include:
- SELECT which fields to be returned – * can be used to indicate all fields.
- FROM which table – databases can have more than one table, each with their own unique name.
- WHERE records meet a condition – LIKE can be used as a wildcard.

Table: world

	name	continent	area	population	gdp	capital
1	Afghanistan	Asia	652230	25500100	20364000000	Kabul
2	Albania	Europe	28748	2821977	12044000000	Tirana
3	Algeria	Africa	2381741	38700000	207021000000	Algiers

SQL in your exams:

- In your exams you will be expected to write a simple SQL query to find a result from a table. The format this will take will be:

SELECT [field names]
FROM [Table name]
WHERE [condition]

Example 1

SELECT population
FROM world
WHERE name = "Albania

world

name	continent	area	population	gdp	capital
Afghanistan	Asia	652230	25500100	20364000000	Kabul
Albania	Europe	28748	2821977	12044000000	Tirana
Algeria	Africa	2381741	38700000	207021000000	Algiers
Andorra	Europe	468	76098	3222000000	Andorra la Vella
Angola	Africa	1246700	19183590	116308000000	Luanda
Antigua and B...	Caribbean	442	86295	1176000000	St. John's
Argentina	South America	2780400	42669500	477028000000	Buenos Aires
Armenia	Eurasia	29743	3017400	9950000000	Yerevan
Australia	Oceania	7692024	23545500	1564419000000	Canberra
Austria	Europe	83871	8504850	394458000000	Vienna
Azerbaijan	Asia	86600	9477100	68727000000	Baku
Bahamas	Caribbean		351461	8043000000	Nassau
Bahrain	Asia	765	1234571	30362000000	Manama

Output from this SQL query:

```
2821977
```

Example 2

SELECT *
FROM world
WHERE name = "Algeria"

world

name	continent	area	population	gdp	capital
Afghanistan	Asia	652230	25500100	20364000000	Kabul
Albania	Europe	28748	2821977	12044000000	Tirana
Algeria	Africa	2381741	38700000	207021000000	Algiers
Andorra	Europe	468	76098	3222000000	Andorra la Vella
Angola	Africa	1246700	19183590	116308000000	Luanda
Antigua and B...	Caribbean	442	86295	1176000000	St. John's
Argentina	South America	2780400	42669500	477028000000	Buenos Aires
Armenia	Eurasia	29743	3017400	9950000000	Yerevan
Australia	Oceania	7692024	23545500	1564419000000	Canberra
Austria	Europe	83871	8504850	394458000000	Vienna
Azerbaijan	Asia	86600	9477100	68727000000	Baku
Bahamas	Caribbean		351461	8043000000	Nassau
Bahrain	Asia	765	1234571	30362000000	Manama

Output from this SQL query:

```
Algeria,Africa,238174,38700000,207021000000,Algiers
```

Example 3

SELECT *
FROM world
WHERE name LIKE "A%" AND population > 10000000
ORDER BY name ASC

world

name	continent	area	population	gdp	capital
Afghanistan	Asia	652230	25500100	20364000000	Kabul
Albania	Europe	28748	2821977	12044000000	Tirana
Algeria	Africa	2381741	38700000	207021000000	Algiers
Andorra	Europe	468	76098	3222000000	Andorra la Vella
Angola	Africa	1246700	19183590	116308000000	Luanda
Antigua and B...	Caribbean	442	86295	1176000000	St. John's
Argentina	South America	2780400	42669500	477028000000	Buenos Aires
Armenia	Eurasia	29743	3017400	9950000000	Yerevan
Australia	Oceania	7692024	23545500	1564419000000	Canberra
Austria	Europe	83871	8504850	394458000000	Vienna
Azerbaijan	Asia	86600	9477100	68727000000	Baku
Bahamas	Caribbean		351461	8043000000	Nassau
Bahrain	Asia	765	1234571	30362000000	Manama

Output from this SQL query:

```
Afghanistan,Asia,652230,25500110,20364000000,Kabul
Algeria,Africa,238174,38700000,207021000000,Algiers
Angola,Africa,1246700,19183590,116308000000,Luanda
Argentina,South America,2780400,42669500,477028000000,Buenos Aires
Australia,Oceania,7692024,2354550,1564419000000,Canberra
```

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
Use of Arrays

Key Terminology	BCS Definition
Array	"A set of data items of the same type grouped together using a single identifier. Each item is addressed by its variable name and a subscript."

What is an array?

- Think of an array as a variable that can contain more than one data item – for example, storing a list of names.
- We can store a list of names by allocating a contiguous part of memory for that data.
  Contiguous means all the data is stored together, one element after the other.
- Note: Lists are different to arrays because they are not contiguous.
- The program will know where our array starts in memory – in this case, address 05.
- It uses an index relative to this start point to allow us to easily access the array's contents.
- For example, "Jane" is at index 2.

RANDOM ACCESS MEMORY

ADDRESS	INSTRUCTION/DATA	
01		
02		
03		
04		index
05	Boris	[0]
06	Marek	[1]
07	Jane	[2]
08	Ryan	[3]
09		
10		
11		
12		

Arrays

- A static number of related data items are stored together in the same memory space.
- Each data item has the same data type.
- A specific data item (element) can be found using its index.
- Arrays usually start at 0 for the first data item, known as zero-indexed.
- Arrays may be single- or multi-dimensional.
- You can visualise these dimensions as a column (one-dimensional) or table (two-dimensional).

Names (One-dimensional array)	
Index	Name
0	Craig
1	Dave
2	Sam
3	Carol

names[2] = "Sam"

Names and scores (Two-dimensional array)		
	Name	Score
Index	0	1
0	Craig	112
1	Dave	75
2	Sam	103
3	Carol	97

names[1][0] = "Dave"

Note: [1][0] could also be "112" depending on how you choose to store the data. It doesn't matter whether you visualise rows or columns first, providing you are consistent in your program. Read exam questions carefully to be sure which way around the array is being stored.

Arrays: OCR Exam Reference Language

- Arrays will be zero-based and declared with the keyword array.

Example one-dimensional array:

```
array names[5]
names[0]="Ahmad"
names[1]="Ben"
names[2]="Catherine"
names[3]="Dana"
names[4]="Elijah"
print(names[3])
```

Concept	Keyword/Symbol
`Declaration`	array names[…]
`Assignment`	names[…] = …

Example two-dimensional array:

```
array gameboard[8,8]
gameboard[0,0]="rook"
print(gameboard[3,4])
```

Concept	Keyword/Symbol
`Declaration`	array gameboard[…,…]
`Assignment`	gameboard[…,…] = …

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
Sub-programs: Procedures and Functions

Key Terminology	BCS Definition
Sub-programs	"A block of code given a unique identifiable name within a program. Supports code reuse and good programming technique."
Procedure	"A block of code within a program that is given a unique, identifiable name. Can take upwards of zero parameters when it is called. Should be designed and written to perform a task or action that is clearly indicated by its name."
Function	"A block of code within a program that is given a unique identifiable name. Can take upwards of zero parameters when it is called and should return a value. Should be designed and written to perform a task or action that is clearly indicated by its name."

What is a sub program?

- Sub-programs are self-contained blocks of code within a larger program that perform specific tasks. They are reusable and can be invoked multiple times, often with inputs (parameters), and may return outputs. Examples include functions, procedures, and methods in programming languages.

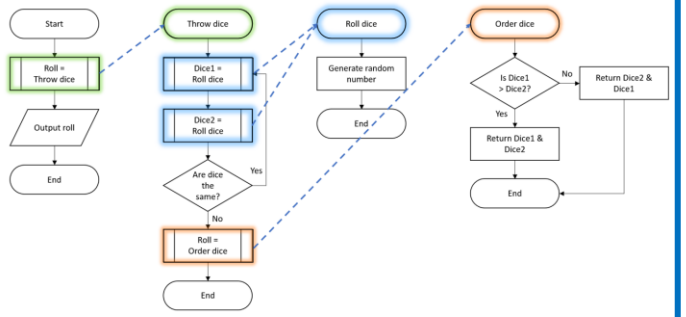
There are two main types of sub-program:

- **Procedures**
  - Carry out a task.
  - Provide structure to code.
- **Functions**
  - Carry out a task and **return** a value.
  - Create reusable program components.

- Larger programs are developed as a set of sub-programs or *sub-programs*.
- Structuring code using sub-programs makes it easier to read and debug.
- Functions return values and create reusable program components.
- Procedures structure a program in a modular way, making it easier to read.
- Each sub-program can be tested independently.
- Sub-programs can be saved in libraries and reused in other programs.

Advantages of using sub-programs

- Programs are easier to write and debug.
- Components can be easily reused.
- Functions can be stored in a library for easy reuse across multiple programs.
- For example, import random imports the random library of functions into a program.
- Programs are also easier to test.



Here we can see how sub-programs are called from inside flowcharts.

Sub-programs – procedures and functions (OCR Exam Reference Language)

Here, the *triple* function takes in one parameter – an integer – and returns that number multiplied by 3.

```
function triple(number)
   return number * 3
endfunction
```

Here, we are calling the function and passing in the

```
y = triple(7)
```

Here, the *greeting* function takes in one parameter – a string – and concatenates it with

```
procedure greeting(name)
   print("hello"+name)
endprocedure
```

Here, we are calling the procedure and passing in the string "Craig".

```
greeting("Craig")
```

Sub-programs

Concept	Keyword/Symbol

Procedure	`procedure name(…)`
	`…`
	`endprocedure`

Calling a procedure	`procedure(parameters)`

Function	`function name(…)`
	`…`
	`return …`
	`endfunction`

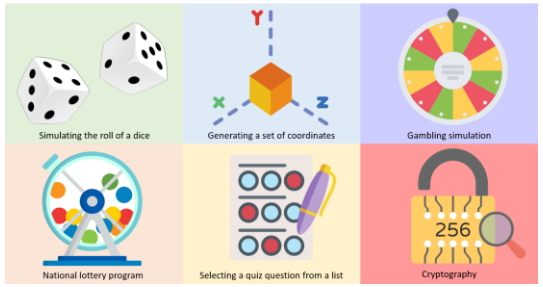
Calling a function	`function(parameters)`

GCSE Computer Science Knowledge Organiser
SLR 2.2 Programming Fundamentals:
Random Number Generation

Key Terminology	BCS Definition
Random number generation	"Most programming languages have built-in functions or libraries that allow you to easily generate random numbers. Creating truly random numbers is actually rather difficult for a computer, and these algorithms are quite complex."

There are many situations when you might want to generate a random number:

- Simulating the roll of a dice
- Generating a set of coordinates
- Gambling simulation
- National lottery program
- Selecting a quiz question from a list
- Cryptography



- Random numbers can be referenced with the following notation:

```
num = random(2,5)
```

Concept	Keyword/Symbol
Random number	random(… , …)

This will generate a random number between 2 and 5 and store it in the variable *num*.

Example code for rolling three dice using a random number generation in Python

```
import random    # Importing the libary files

# Generating a random integer using randint to get a
number between 1 and 6 for three dice
dice1 = random.randint(1,6)
dice2 = random.randint(1,6)
dice3 = random.randint(1,6)

# Displaying the rollded dice numbers
print("Dice rolled:",dice1,dice2,dice3)

# Checking which dice match and calculating the score
if dice1 == dice2 and dice1 == dice3:
    score = dice1 + dice2 + dice3
elif dice1 == dice2:
    score = dice1 + dice2 - dice3
elif dice1 == dice3:
    score = dice1 + dice3 - dice2
elif dice2 == dice3:
    score = dice2 + dice3 - dice1
else:
    score = 0

# Outputting the score
print("Score: ",score)
```