OCR Computer Science GCSE

2.2 – Programming fundamentals

Advanced Notes

2.2.1 Programming fundamentals

Core Programming Statements

Statements	Description	Examples (pseudocode)
Variable Declaration	Creates a variable to store data.	Example: `name = "Alex"`
Constant Declaration	A value that does not change while the program runs. Often given fully uppercase identifiers.	Example: `PI = 3.14`
Assignment	Setting or updating a value in a variable.	Example: `score = score + 10`
Input	Receiving data from the user.	Example: `name = input("Enter your name")`
Output	Displaying data or information to the user.	Example: `output("Hello World")`

Input and output statements may vary depending on if your code is written in program code (e.g. Python) or pseudocode. For example, output in Python would be done with '`print()`', whereas in pseudocode it is done with '`output()`'.

Programming Constructs

There are three constructs used in algorithms, which help to make them structured, easy to understand, and control the flow of the program:

1. Sequence – instructions executed in order
2. Selection – decisions (`IF`, `ELSE`)
3. Iteration – repetition (`WHILE`, `FOR`)

What Are Arithmetic Operations?

Arithmetic operations are the basic mathematical calculations that can be performed in a programming language. These are essential for processing numerical data in algorithms and programs.

Standard Arithmetic Operators

Operation	Symbol	Example	Result
Addition	+	3 + 2	5
Subtraction	-	7 - 4	3
Multiplication	*	5 * 3	15
Real Division	/	10 / 4	2.5
Modulus (MOD)	%	5 % 2	1
Quotient (DIV)	//	8 // 3	2
Exponentiation	^	4^2	16

Modulus returns the remainder of a division, for example, 7 % 4 would return 3.

Quotient returns the largest integer from a division result, for example, 11 // 2 would return 5.

Modulus is useful as it can be used to identify if a number is even or odd, for example:

- 12 % 2 = 0 (even)
- 13 % 2 = 1 (odd)

An odd number modulus 2 will always have a remainder of 1, whilst an even number has no remainder.

Modulus and quotient division can be used to return both the integer and decimal parts of a number after division. The full result can be obtained by adding the integer and decimal, which is formed by dividing the remainder from the modulus division by the divisor.

What Are Comparison Operations?

Comparison operations are used to compare different items of data, and either return `True` or `False`.

Standard Comparison Operators

Operation	Symbol	Example	Result
Equal to	`==`	`5 == 5`	`True`
Not equal to	`!=`	`3 != 4`	`True`
Less than	`<`	`2 < 5`	`True`
Greater than	`>`	`6 > 7`	`False`
Less than or equal to	`<=`	`5 <= 5`	`True`
Greater than or equal to	`>=`	`7 >= 10`	`False`

What Are Boolean Operations?

Boolean operations are logical operators that work with Boolean values (`True` or `False`). They are used in conditions to control the flow of programs.

Boolean Operators

Operator	Description	Example	Result
`NOT`	Reverses the Boolean value.	`NOT True`	`False`
`AND`	Returns `True` if both input conditions are true.	`True AND True`	`True`
`OR`	Returns `True` if either input condition is true.	`True OR False`	`True`

Boolean operators can be combined in complex logic, such as:

```
if age > 18 AND height >= 155:
    allowEntry
```

2.2.2 Data types

What Are Data Types?

In programming, a **data type** defines the kind of data a variable or constant can hold. It tells the program how the data will be **stored, processed, and displayed** - including which operations can be performed on it.

Common Data Types

Term	Description	Examples
Integer (int)	Whole numbers only, no decimals.	`5`, `-20`, `0`
Real (float)	Numbers that include a fractional/decimal part. Also called float in some languages.	`3.14`, `-0.5`, `99.99`
Boolean (bool)	Often used for conditions and logic.	Only has two values: `True` or `False`
Character (char)	A single symbol or letter.	Must be enclosed in quotation marks (e.g., `'A'`, `'5'`, `'#'`)
String (str)	A sequence of characters.	Examples: `"Hello"`, `"123"`, `"£$%"`

Why Data Types Matter

- They help the computer understand how to store and manipulate data.
    - For example, 123 is an integer, whereas “123” is a string – they are different data types and so can undergo different operations.
- Choosing the right data type ensures the program runs efficiently and without errors.
- Some operations are only valid for certain types (e.g. you can’t divide strings).

Casting Data Types

Casting refers to changing data types of a variable, for example from integer to string:

- `stringNumber = str(123)`

2.2.3 Additional programming techniques

What Is String Manipulation?

String manipulation refers to measuring the length of strings, concatenating strings (joining strings together) or slicing strings (extracting substrings).

Key String Operations

Operation	Description	Example
`length`	Returns the number of characters in a string.	`name = "Alice"` `name.length() → 5`

`subString`	Extracts a sequence of characters within a string.	`greet = "Hello World"`
		`greet.subString(0, 5) → "Hello"`
		Note: first argument is the starting position, second argument is the number of characters.

`concatenation`	Joins strings together.	`"Hi" + " there" → "Hi there"`

What Is File Handling?

File handling refers to the reading or writing to or from an external file.

Files must be opened in either 'read' or 'write' mode and closed after all file operations have been performed.

File Handling Operations

Operation	Code	Text File (sample.txt)

Reading from a file	`myFile = openRead("sample.txt")`
	`x = myFile.readLine()`
	`print(x)` **Note: this prints "Good Morning!"**
	`myFile.close()`
		Good Morning!

Writing to a file	`myFile = openWrite("sample.txt")`
	`myFile.writeLine("Goodbye")`
	`myFile.close()`
		Goodbye

Note: a file *must* be closed after an operation has been performed on it using myFile.close(). If not, the changes and edits made to the file may not be saved.

What Is a Data Structure?

A data structure is a way of organising and storing related data so it can be used efficiently in a program. It helps manage collections of data.

1. Records

What is a record?

- A data structure used to **group different types of data** under one structure.
- Each field in a record can have a **different data type**.
- Similar to a **row in a database table**.

Example:

```
plaintext

RECORD Car
  make : String
  model : String
  reg : String
  price : Real
  noOfDoors : Integer
ENDRECORD
```

2. Arrays

What is an array?

- A fixed-size data structure with a **collection of similar data items** (elements) stored under a **single name**.
- Each item is accessed using an **index** (position number).

Characteristics:

- Items must be of the **same data type**.
- Indexing starts at **0.**
- They are **static** (cannot change size/length during run-time)

One-Dimensional Array (1D)

- A **single list** of items.
- Example:
  `scores = [10, 20, 30]`
  `scores[1]` → `20`

Two-Dimensional Array (2D)

- An array of arrays (like a **database table or grid**).
- Example:

```
seating = [
  ["Alice", "Bob"],
  ["Cara", "Dan"]
]
```

To access a value in a 2D array, you need to use two indexes:
- The first index selects the row (the sub-array)
- The second selects the column (item within that sub-array).

For example: to return `"Bob"` you would need to write `seating[0][1]`.
- 0 selects the first array (row): [“Alice”, “Bob”]
- 1 selects the second item within that array: “Bob”

You can think of a 2D array like a table or grid where:
- Each row is a separate array
- Each column is an item within those arrays

		Column	
		0	1
Row	0	“Alice”	“Bob”
	1	“Cara”	“Dan”

What Is a Database?

A database is used to store large amounts of data into organised tables. Each column will have a heading to define a field (column), for example:

employeeNo	firstName	lastName	dateOfBirth
0013	Bob	Smith	28/01/2001

In the example above, each row (record) will have a **unique value** to identify that row, in this case it would be the employeeNo which is unique to every employee.

What Is Structured Query Language (SQL)?

SQL is a language used to search for, manage, and manipulate data in a database.

There are three main SQL commands: **SELECT**, **FROM** and **WHERE**.

- The **SELECT** command is used to retrieve information about a particular field in a database
- The **FROM** command refers to the database being searched
- The **WHERE** command is used to apply conditions to a search

Table: Cars

Model	Manufacturer	Price	Year	Sold
`Polo`	Volkswagen	4995	2010	TRUE
`i10`	Hyundai	5225	2013	FALSE
`Fiesta`	Ford	3995	2009	TRUE

SQL commands take the following form:

```
SELECT <field>
FROM <tablename>
WHERE <condition>
```

Example using the table above:

- ```
  SELECT Model
  FROM Cars
  WHERE Sold = FALSE
  ```

  >> i10

You can also retrieve multiple fields at once, or apply multiple conditions, for example:

- ```
  SELECT Model, Manufacturer, Price
  FROM Cars
  WHERE Sold = True and Price > 4000
  ```

  >> Polo, Volkswagen, 4995

The SQL wildcard *, can be used to represent all fields in the SELECT command:

- ```
  SELECT *
  FROM Cars
  WHERE Manufacturer = “Ford”
  ```

  >> Fiesta, Ford, 3995, 2009, True
  ```

What is a Subprogram?

A subprogram is a section of code which performs a specific task, and can be called whenever it is needed to carry out that task. They are also called sub-routines.

There are two main types of subprograms:

- **Functions:** a type of subprogram that performs a task and **returns a value.** For example:

  ```
  function density(mass, volume)
      d = mass / volume
      return d
  endfunction
  ```

- **Procedures:** a type of subprogram that performs a task and **does not return a value.**

  ```
  procedure greeting()
      print("Hello World")
  endfunction
  ```

The variables which are found in the brackets of a subprogram are known as **parameters**. These are values which can be passed into a subprogram when it is called, as in the function above, 'mass' and 'volume' are both parameters.

To use a subprogram, you must call it in the main program, for example:

```
function density(mass, volume)
    d = mass / volume
    return d
endfunction

// Main program
userMass = input("Enter mass")
userVolume = input("Enter volume")
userDensity = density(userMass, userVolume) ← The function is called here
```

Local and Global Variables

Local variables are those which are defined inside of a subprogram, and can only be used inside of that subprogram. Local variables are overwritten in/removed from memory when the subprogram they were created in ends.

Global variables are those which are defined in the main program and can be used anywhere, including all subprograms and the main program.

Example:

```
function density(mass, volume)
    d = mass / volume ← 'd' is a local variable and can only be used in this function
    return d
endfunction

// Main program
userMass = input("Enter mass") ← 'userMass' is a global variable
userVolume = input("Enter volume")
userDensity = density(userMass, userVolume)

print(d) ← If we add this line in the main program we would get an error because 'd' does not exist in the main program, only in the 'density' function.
```

Why use Subprograms?

Subprograms are incredibly useful because they:

- Make code more structured and modular
- Allows code to be reused, without having to be repeated or rewritten
- Make programs easier to test and maintain; a subprogram only has to be tested once
- Allows shared workload by splitting the development of subprograms across a team

What is Random Number Generation?

Random number generation is the ability to produce unpredictable numeric values within a specified range. It can be used for a wide range of purposes, such as games (e.g. dice throws), testing, or security (passwords).

Example (pseudocode):

```
num = RANDOM_INT(0, 6)
```

Example (Python):

```
num = random.randint(0, 6)
```

Note: the random.randint function in python is inclusive, meaning the numbers generated could be the ones passed as parameters. In the above example this would include 0 and 6.

Typical Uses

- Rolling a die
- Picking a random question or card
- Generating test values for simulations
- Randomly deciding outcomes in games (e.g. attack chance)

Good Practice

- Store the random value in a variable if it will be used more than once
- Combine with loops or conditions for more dynamic outcomes