AQA Computer Science GCSE

3.2 Programming

Advanced Notes

3.2.1 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**.

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	`True` or `False`
Character (char)	A single symbol or letter, enclosed in single quotes for most programming languages	`'A'`, `'a'`, `'#'`
String (str)	A sequence of characters, enclosed in double quotes for most programming languages	`"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).

3.2.2 Programming concepts

What Are Programming Concepts?

Programming concepts are the building blocks of writing code. They include the fundamental statements, control structures, and organisational techniques used to create functioning programs.

Core Programming Statements

Statements	Description	Examples
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`

Iteration (Loops)

Definite (Count-Controlled) Iteration

- Repeats a fixed number of times.
- Example: `FOR i ← 1 TO 5`

Indefinite (Condition-Controlled) Iteration

- Repeats **while or until** a condition is met.
- Examples:
    - `WHILE notDone`
    - `REPEAT ... UNTIL done`

Nested Iteration

- A loop **inside another loop**.
- Example:

```css
WHILE notSolved
  FOR i ← 1 TO 5
    ...
```

Selection (Decision-Making)

IF Statements

- Executes code **only if a condition is true**.
- Can include `IF`, `ELSE IF`, and `ELSE`.

Nested Selection

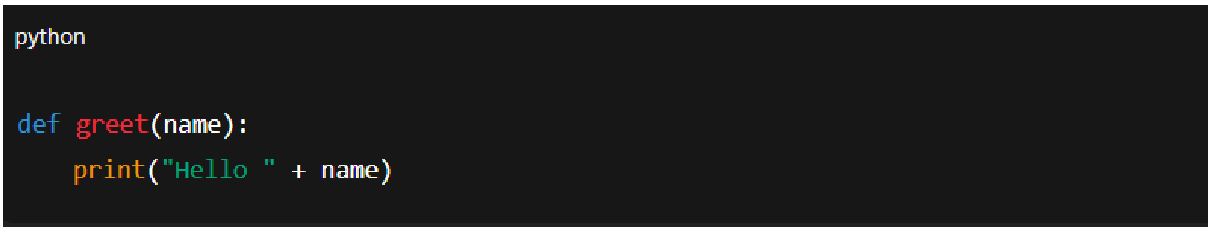
- An `IF` statement **inside another `IF`**.
- Example:

```markdown
IF passed THEN
  IF grade > 90 THEN
    ...
```

Subroutines (Procedures/Functions)

Subroutines

- A block of code given a **unique name** that can be called multiple times.
- May include **parameters** and return **values**.
- Example:



python

```
def greet(name):
    print("Hello " + name)
```

Meaningful Identifier Names

- Use clear, descriptive names for variables, constants, and subroutines.
- Good naming improves **readability** and **understanding** of code.
- Example: Use `totalMarks` instead of `x`.

Note: Subroutines are covered in more detail in What Is a Subroutine?

3.2.3 Arithmetic operations

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`

Integer Division and Remainder

These are used when working with **whole numbers only**.

Operation	Description	Example	Result
Integer Division	Gives the **whole number quotient**	`11 DIV 2`	`5`
Modulus (Remainder)	Gives the **remainder**	`11 MOD 2`	`1`

These two together **completely describe a division** with remainder.

Modulus can also be performed using its sign, e.g. 11 % 2 is the same as 11 MOD 2.

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

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

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

MOD can also be used similarly to check whether a number is a multiple of another.

3.2.4 Relational operations

What Are Relational Operations?

Relational operations are used to compare two values. They return a Boolean value:

- `True` if the comparison is correct
- `False` if it is not

These operations are commonly used in conditions, such as `IF` statements and loops.

Relational Operators

Operation	Symbol in most languages	Example	Result
Equal to	`==`	`5 == 5`	`True`
Not equal to	`!=` or `<>`	`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`

Note: The actual symbol may vary slightly between programming languages.

In Python: `==`, `!=`

In VB.NET: `=`, `<>`

Where Are These Used?

- **In `IF`, `ELSE IF`, and `WHILE` statements**
- To make decisions in code based on **comparisons**
- To control the flow of **loops** and **branches**

3.2.5 Boolean operations

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 Explained

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

Combined Conditions

Boolean operators can be combined in complex logic:

```
python

IF age > 18 AND hasID THEN
  allowEntry
```

```
python

WHILE NOT finished:
  ...
```

Truth Tables

AND

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

OR

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False

NOT

A	NOT A
True	False
False	True

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. Arrays

What is an array?

- 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 usually starts at **0** (in most languages).

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 **table or grid**).
- Example:

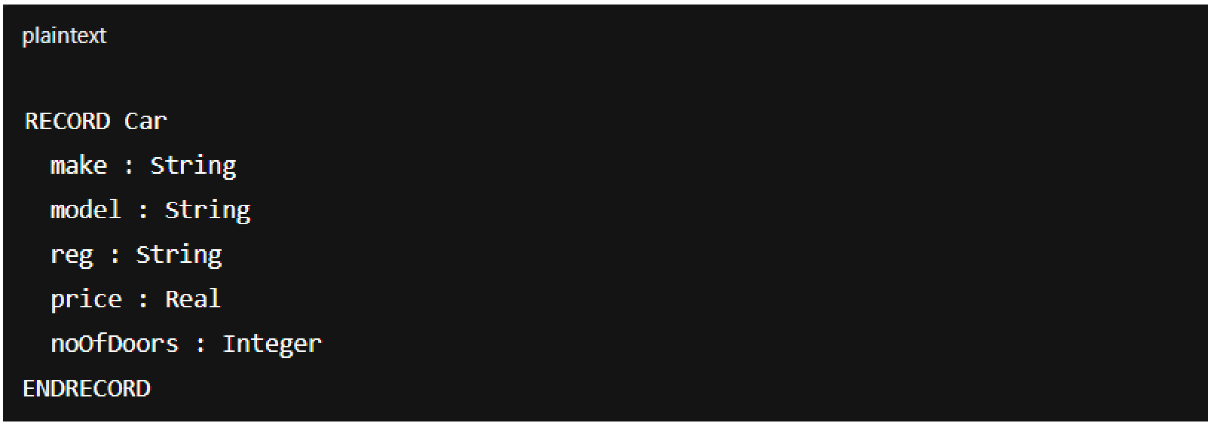
```ini
seating = [
  ["Alice", "Bob"],
  ["Cara", "Dan"]
]
```

2. 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, defined 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
```

Arrays vs Records

Feature	Arrays	Records
Data Types	All elements must be the same	Can contain different data types
Accessed by	Index	Field name
Suited for	Lists of similar items	Grouping related attributes

3.2.7 Input/Output

What Is Input/Output in Programming?

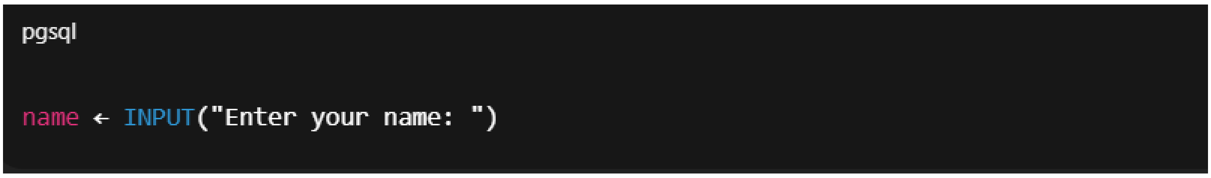
Input/Output (I/O) refers to how a program interacts with the outside world — specifically how it:

- Receives data from the user (input)
- Displays data or information to the user (output)

Input (Getting Data from the User)

Used to collect data that a program needs to process. Typically stored in a variable after being entered.

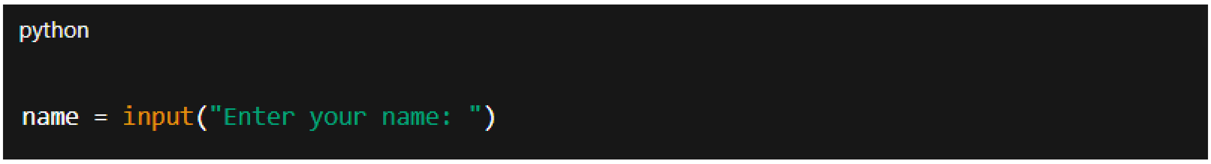
Example (Pseudocode):



pgsql

```
name ← INPUT("Enter your name: ")
```

Example (Python):



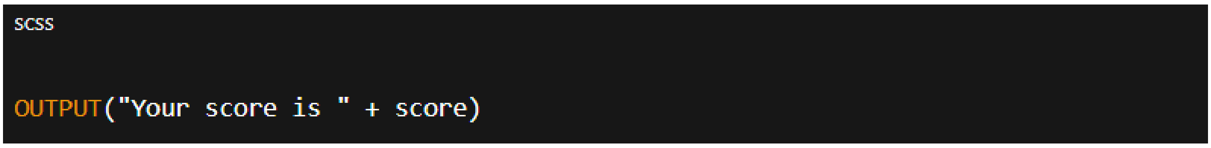
python

```
name = input("Enter your name: ")
```

Output (Displaying Data)

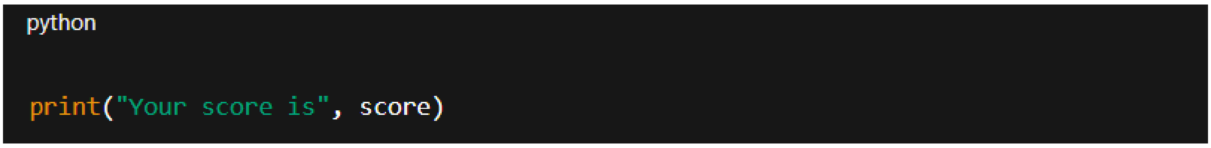
Used to show messages, results, or prompts to the user.

Example (Pseudocode):



```scss
OUTPUT("Your score is " + score)
```

Example (Python):



```python
print("Your score is", score)
```

Notes:

- Outputs can display text, numbers, or variable values.
- Inputs are usually strings by default and may need type conversion (e.g., `int(input(...))` in Python).
- Input/output operations are often used with selection and iteration, such as prompting repeatedly until valid data is entered.

3.2.8 String handling

What Is String Handling?

String handling refers to the operations you can perform on strings (text data). Such as measuring length, extracting substrings, combining strings, or converting between types.

A string is a sequence of characters, e.g. `"hello123!"`

Key String Operations

Operation	Description	Example
`length`	Returns the number of characters in a string	`length("hello") → 5`
`position`	Returns the index of a character or substring	`position("hello", "e") → 1`
`substring`	Extracts a sequence of characters within a string	`substring("computer", 0, 2) → "com"`
`concatenation`	Joins strings together	`"Hi" + " there" → "Hi there"`

Character ↔ Code Conversions

Task	Function	Example
Character → ASCII code	`ASC("A")`	Returns `65`
ASCII code → Character	`CHR(65)`	Returns `"A"`

String Conversion Operations

Task	Function (in pseudocode/Python style)	Example
String → Integer	`int("42")`	`"42"` → `42`
String → Real	`float("3.14")`	`"3.14"` → `3.14`
Integer → String	`str(42)`	`42` → `"42"`
Real → String	`str(3.14)`	`3.14` → `"3.14"`

3.2.9 Random number generation

What Is Random Number Generation?

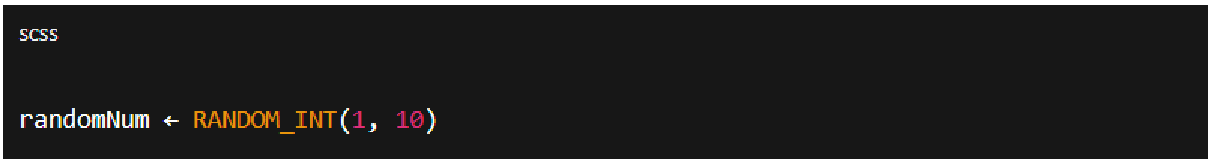
Random number generation is the ability to produce unpredictable numeric values within a specified range. It is commonly used in programs that involve:

- Games
- Simulations
- Testing
- Security (e.g. simple password generators)

Key Features

- Produces a different (seemingly random) value each time it runs
- Usually requires you to define a **range** (minimum and maximum)
- Must be assigned to a **variable** for use

Example Syntax (Pseudocode):

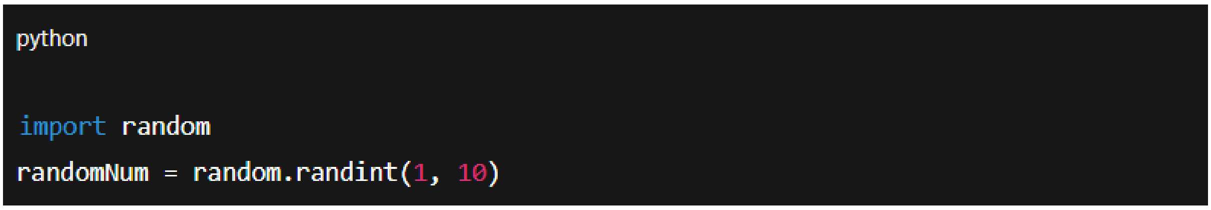


```
scss

randomNum ← RANDOM_INT(1, 10)
```

This assigns a random integer between 1 and 10 to the variable `randomNum`.

Example in Python:



```python
python

import random
randomNum = random.randint(1, 10)
```

Note: in Python, random.randint is inclusive of the parameters. For the Python example above, 1 or 10 could be assigned to randomNum.

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

3.2.10 Structured Programming and Subroutines

What Is Structured Programming?

Structured programming is a method of writing clear, modular, and easy-to-understand code using three core principles:

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

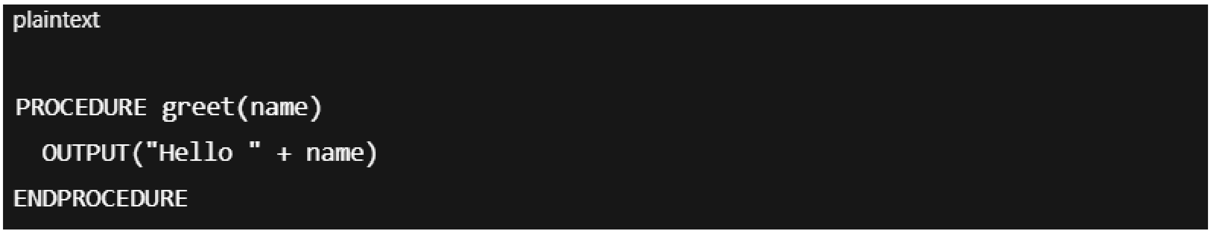
What Is a Subroutine?

A **subroutine** is a block of code that performs a **specific task** and can be **reused** by calling it by name.

Types:

- **Procedure**: performs an action (may or may not return a value)
- **Function**: performs an action and **returns a value**

Example (Pseudocode):



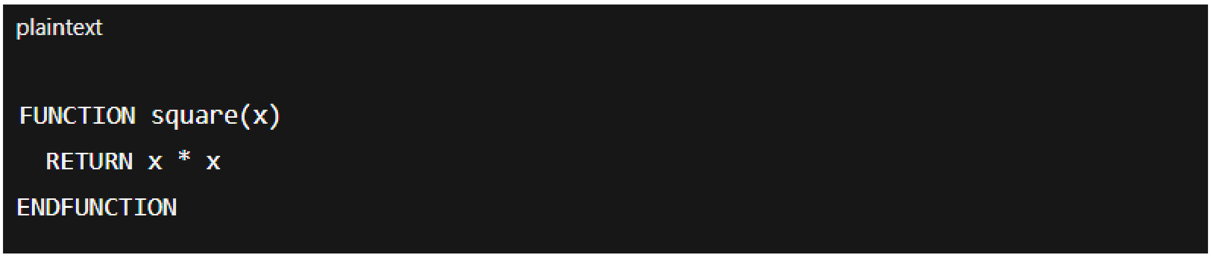
plaintext

```
PROCEDURE greet(name)
  OUTPUT("Hello " + name)
ENDPROCEDURE
```

Parameters and Return Values

- **Parameters** allow data to be passed into a subroutine
- **Return values** allow data to be passed back to the main program
  - The returned value should be assigned to a variable in the main program

Example:



plaintext

```
FUNCTION square(x)
  RETURN x * x
ENDFUNCTION
```

Local Variables

- Declared **inside** a subroutine
- Can only be accessed **within** that subroutine
- Prevents conflicts between subroutines
- Only exist while the subroutine is executing

Advantages of Using Subroutines

Benefit	Explanation
Modularity	Code is split into manageable parts
Reusability	Same subroutine can be reused without rewriting code
Readability	Code is easier to understand and maintain
Testing and Debugging	Subroutines can be tested independently
Reduced Repetition	Avoids duplicating blocks of logic
Split Workload	Subroutines can be spread amongst a team to complete

Structured Approach Summary

Feature	Benefit
Sequence	Code runs in clear logical order
Selection	Makes decisions based on conditions
Iteration	Handles repetitive tasks
Modular design	Easier maintenance and collaboration

3.2.11 Robust and secure Programming

What Is Robust and Secure Programming?

Robust and secure programming is about writing code that:

- Prevents crashes
- Handles incorrect input
- Protects user data
- Deals with errors effectively

It ensures that programs are safe, reliable, and resistant to failure.

1. Data Validation

Validation ensures that input is sensible before it’s processed.

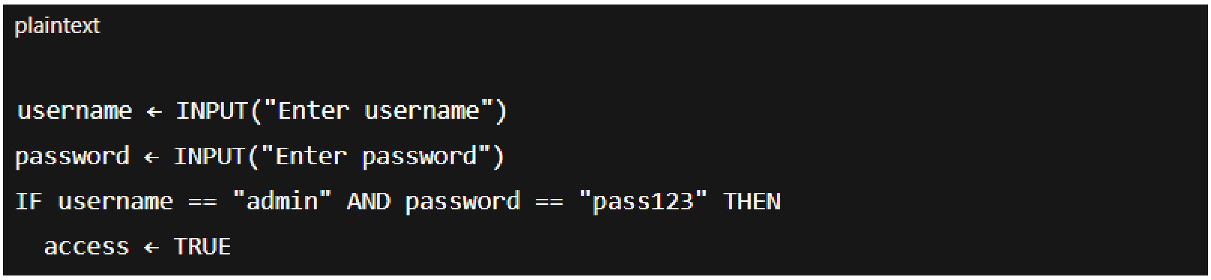
Common Checks:

Type	Description	Example
Length check	Input must be a minimum/maximum length	Name must be at least 2 characters
Presence check	Input cannot be left blank	Email cannot be empty
Range check	Number must fall within specific range	Age must be between 1–120

2. Authentication

Authentication checks if a user is who they claim to be.

Example:



plaintext

```
username ← INPUT("Enter username")
password ← INPUT("Enter password")
IF username == "admin" AND password == "pass123" THEN
  access ← TRUE
```

Note: Plain text is fine for GCSE – encryption is not required.

3. Testing and Test Data Types

Testing is used to:

- Check if a program **works as intended**
- Find and fix **bugs or errors**

Types of Test Data:

Type	Purpose	Example (Range: 1–10)
Normal	Typical input	`5`
Boundary	On the edge of valid range	`1 and 10`
Erroneous	Invalid input	`-1`, `eleven`, `"abc"`

4. Types of Errors

Type	Description	Example
Syntax Error	Breaks the rules of the language (won't run)	Missing colon in Python
Logic Error	Code runs but produces the wrong result (harder to spot)	Using `+` instead of `*`

Selecting Suitable Test Data

You should be able to:

- Identify appropriate test data for a given input field
- Justify why it's used (e.g. "Boundary data ensures edge cases work")