Authentication

- A coding method to check that:
  - The user is who they say they are
  - The user is allowed to access the program
- Can be as simple as asking for a username and password
- There are three main authentication factors
  - Something you are, such as a fingerprint or iris scan.
  - Something you know, such as a password, pin or secret answer to a question.
  - Something you have such as a swipe card or mobile phone app.
- 2 factor authentication is where two different authentication types are required to access the program.

Logic Errors

- An error in the way the program works, causing it to not do what it should.
- May be the incorrect use of operators such as entering < instead of >
- May be the creation of an infinite loop.
- May be the accidental reuse of a variable name
- A program will run with logic errors but will not function correctly.

Syntax Errors

- A mistake in how the code is written, breaking the rules of the programming language.
- May be a misspelling or typo such as prnit instead of print.
- May be a missed bracket.
- May be using a variable without declaring it.
- A program will not run if there are syntax errors.

Input Validation

- Any user inputs may be incorrect, the program be able to handle this.
- Validation applies rules to inputs, data which does not follow the rules is rejected to prevent it from crashing the program.
  - Range Check – the input must be within a range. Usually applied to numbers and dates. For example, when inputting the required quantity into an order form, the number must be greater than 0 and less than the total stock available.
  - Length Check - the input must not be too long or too short. For example, a password must be at least eight characters, but not more than 30.
  - Presence Check – the input must be present. For example, requiring a credit card number for an online order
  - Format Check - the data must be in a specific format. For example, an email address must have an @ symbol and at least one dot.
  - Type Check - the data must be a specific type, such as requiring a currency input to be only numbers.
- Validation will not catch all errors as users may still make typos.
- Verification requires the user to enter key info twice to reduce the risk of this.

2.3 – Producing Robust Programs

Good Practice

```
VAR Password as String
VAR User as String
Password=Input("Enter the password")
#Ensure the password is correct
IF Password="letmein" THEN
      #Apply access levels
      IF User="Technician" THEN
            Allow Unrestricted Access
      ELSE
            Allow Restricted Access
ELSE
      Deny Access
END IF
```

Bad Practice

```
VAR X as String
VAR Y as String
X=Input("Enter the Password")
IF X="letmein" THEN
IF Y="Technician" THEN
Allow Unrestricted Access
ELSE
Allow Restricted Access
ELSE
Deny Access
END IF
```

Naming Conventions

- Using the same rules for naming throughout the program make it easier to read.
- These are applied to variables, functions, procedures, etc.
- Should be easy to read.
- Should be meaningful.
- Makes the code easier to read and to understand.
- For Example, FirstName is a better variable name than just X or FN.

Test Plans

- Provides structure to testing.
- Records the result of testing.
- Should include:
  - The test number
  - The data entered
  - The type of data
  - The expected outcome
  - The result of the test
  - Any action required as a result

Refining Algorithms

- User prompts should be helpful and explain any input validation rules.
- Code should convert inputs to the required data type if needed.
- Loops can be used to request the user re-enter data if it is invalid.
- There may be a limit on how many times the user is asked in the case of passwords or other security fields.

Selecting and Using Suitable Test Data

- A range of data should be used when testing.
- Normal data is correct and what would usually be inputted by the user.
- Boundary data is correct but is the largest or smallest value which a user might input. For example, entering an age of 105.
- Invalid data is too large or small, for example entering an age of 2978.
- Erroneous data is completely incorrect, for example entering Bob into an age field.

Indentation

- Allows code within a particular function or procedure to be grouped together.
- Often used with IF statements.
- Multiple levels of indentation may be used.
- Makes the code easier to read and understand.
- Makes it easier to focus on particular parts of the code when needed.

Commenting

- Lines within the code which are not executed.
- Starts with a certain character depending on the language used. Common symbols include # */ and /
- Informs the reader about bugs or issues in the code
- Explains the functionality of particular code
- Explains the purpose of particular code
- Prevent code from executing without deleting it completely.

Exam Style Question

Explain, using examples, way to improve the maintainability of the program shown above [4]

Indent the lines within the IF statement in order to make the code easier to read.
Use sensible variable names, for example ‘X’ could instead be called ‘Password’ and ‘Y’ could be called ‘User’. This would make the program easier to read.

Anticipating Misuse

- May be a brute force attack on the program.
- May be a user entering an incorrect input to try and break the program.
- May be a user entering code into input fields to access parts of the program they should not.
- May simply be an error in input.

Sub Programs

- Procedures carry out a set of instructions and do no not return a value.
- Functions are similar but do will return a value.
- Both procedures and functions can accept parameters
- Parameters are values passed into a sub program. These are referred to as arguments when calling the sub program
- They provide structure to the code.
- They make code easier to understand.
- They allow code to be easily reused.
- They allow the program to be shorter as code need not be written out multiple times.

Testing

- Newly written code often contains errors.
- Testing helps to locate and remove these errors.
- Testing ensures the program works in the way it should.

Iterative Testing

- Takes place whilst the program is being written.
- The programmer tests individual lines or sections of code as they are written.
- If an error is found, it is fixed and the code tested again.
- This process repeats, or iterates, until the code works as intended.
- It is easier to fix errors in smaller sections of code.

Final Testing

- Takes place once the code is finished.
- A final check to make sure the code works correctly.
- Makes sure the program does what it should.
- It can be harder to locate and fix errors at this stage because of the amount of code.