Questions matter

Teaching guide: Data validation and authentication

This resource will help with understanding robust and secure programming. It supports elements of section 3.2.11 of our GCSE Computer Science specification (8525). The guide is designed to address the following learning aims:

- Recognise the inherent unreliability of data entered by a user.
- Investigate ways to validate data using code.
- Be able to write simple authentication routines.

Validation

When writing programs that involve user input, a programmer should assume that the user will make mistakes when entering data and write their programs accordingly.

Code can be written that validates the user's input before carrying on. Adding validation to program code can help to reduce the likelihood of incorrect user input causing a program to crash or produce incorrect output, thereby making a program more robust.

Students should be aware that validation only checks that input data is feasible, such as being in the correct format. It does not check that the data is correct. For example, validation could not detect the input of a test percentage of 56 when a student scored 65 as both are feasible marks, but it could reject a mark of 566 which is not feasible.

Different types of validation

For GCSE, students need to be able to perform simple validation checks such as:

- length check: checking if an entered string has a minimum length
- presence check: checking that a string is not empty
- range check: checking if numeric data entered lies within a given range (e.g. a month between 1 and 12).

For GCSE, students do **not** need to be able to check that data is of the correct type. In exam questions, students can assume that any errors relating to user input will not relate to data type.

Contents

Version 2.0
September 2025

Length checks

A length check typically checks that an entered value contains a certain minimum number of characters but could also check that the value does not exceed a maximum number of characters.

Validation is usually implemented using **indefinite iteration**. This can be achieved using a **pre-conditioned** or **post-conditioned loop**. The table below shows pseudo-code for a length check using post-conditioned and pre-conditioned loops.

Good validation checks will output a message to inform the user if their input has been rejected, but this is not always required (and might not be in an exam question that has a low number of marks allocated to it).

Pseudo-code for simple length check without error message	
Post-conditioned loop	Pre-conditioned loop

`REPEAT`
`  OUTPUT 'Enter password'`
`  password ← USERINPUT`
`UNTIL LEN(password) ≥ 8`
	`OUTPUT 'Enter password'`
	`password ← USERINPUT`
	`WHILE LEN(password) < 8`
	`  OUTPUT 'Enter password'`
	`  password ← USERINPUT`

Pseudo-code for simple length check with error message	
Post-conditioned loop	Pre-conditioned loop

`REPEAT`
`  OUTPUT 'Enter password:'`
`  password ← USERINPUT`
`  IF LEN(password) < 8 THEN`
`  OUTPUT 'Invalid input'`
`  ENDIF`
`UNTIL LEN(password) ≥ 8`
	`OUTPUT 'Enter password:'`
	`password ← USERINPUT`
	`WHILE LEN(password) < 8`
	`  OUTPUT 'Invalid input'`
	`  OUTPUT 'Enter password:'`
	`  password ← USERINPUT`

Using a post-conditioned loop generally requires less code when an error message does not need to be output, but the amount of code required is similar whichever approach is taken when an error message is displayed as part of the validation check. C# and VB.NET support both pre-conditioned and post-conditioned loops but Python only supports pre-conditioned loops.

Examples of length checks implemented in C#, Python and VB.NET.

C#

```
string password;
do
{
  Console.Write("Enter password:");
  password = Console.ReadLine();
  if (password.Length < 8)
  {
    Console.WriteLine("Invalid input");
  }
} while (password.Length < 8);
```

Python

```
password = input("Enter password:")
while len(password) < 8:
  print("Invalid input")
  password = input("Enter password:")
```

VB.NET

```
Dim password As String
Do
  Console.Write("Enter password:")
  password = Console.ReadLine()
  If (password.Length < 8) Then
    Console.WriteLine("Invalid input")
  End If
Loop Until password.Length >= 8
```

Range checks

A range check is used to ensure that an input numeric value falls between two boundaries. For example, a month is between 1 and 12 or a percentage mark is between 1 and 100.

The following pseudo-code shows one way of implementing a range check for a month using a post-conditioned loop.

```
REPEAT
  OUTPUT 'Enter month'
  month ← STRING_TO_INT(USERINPUT)
  IF month < 1 OR month > 12 THEN
    OUTPUT 'Month must be between 1 and 12'
  ENDIF
UNTIL month ≥ 1 AND month ≤ 12
```

To avoid the need to enter similar conditions multiple times (and the potential for programmer error if two sets of conditions are not equivalent), it can be useful to use a Boolean variable to control the termination of a loop, as shown below:

```
REPEAT
  OUTPUT 'Enter month'
  month ← STRING_TO_INT(USERINPUT)
  IF month < 1 OR month > 12 THEN
    OUTPUT 'Month must be between 1 and 12'
    valid ← False
  ELSE
    valid ← True
  ENDIF
UNTIL valid = True
```

Examples of range checks implemented in C#, Python and VB.NET.

C#

```
bool valid;
int month;
do
{
  Console.Write("Enter month:");
  month = Convert.ToInt32(Console.ReadLine());
  if (month < 1 || month > 12)
  {
    Console.WriteLine("Month must be between 1 and 12");
    valid = false;
  }
  else
  {
    valid = true;
  }
} while (valid == false);
```

Python

```
month = int(input("Enter month:"))
while month < 1 or month > 12:
  print("Month must be between 1 and 12")
  month = int(input("Enter month:"))
```

VB.NET

```
Dim month As Integer
Console.Write("Enter month:")
month = Convert.ToInt32(Console.ReadLine())
Do While month < 1 Or month > 12
  Console.WriteLine("Month must be between 1 and 12")
  Console.Write("Enter month:")
  month = Convert.ToInt32(Console.ReadLine())
Loop
```

Use of subroutines for validation

As validation checks are required frequently, implementation of them is a good opportunity to demonstrate the usefulness of subroutines and parameters in facilitating code reuse, once these topics have been covered.

The examples below show how a general-purpose range check could be implemented as a subroutine in C#, Python and VB.NET.

C#

```
static int InputWithRangeCheck(string message, int lowerLimit,
int upperLimit)
{
  int value;
  Console.Write(message);
  value = Convert.ToInt32(Console.ReadLine());
  while (value < lowerLimit || value > upperLimit)
  {
    Console.WriteLine("Must be between " + lowerLimit + " and "
+ upperLimit);
    Console.Write(message);
    value = Convert.ToInt32(Console.ReadLine());
  }
  return value;
}
```

Python

```
def inputwithrangecheck(message, lowerlimit, upperlimit):
  value = int(input(message))
  while (value < lowerlimit or value > upperlimit):
    print("Must be between " , lowerlimit , " and " ,
upperlimit)
    value = int(input(message))
  return value
```

VB.NET

```
Function InputWithRangeCheck(message As String, lowerLimit As
Integer, upperLimit As Integer) As Integer
  Dim value As Integer
  Console.Write(message)
  value = Convert.ToInt32(Console.ReadLine())
  Do While value < lowerLimit Or value > upperLimit
    Console.WriteLine("Must be between " & lowerLimit & " and "
 & upperLimit)
    Console.Write(message)
    value = Convert.ToInt32(Console.ReadLine())
  Loop
  Return value
```

```
End Function
```

Suggested student activities

- Give students a range of scenarios and ask them to choose the most appropriate validation checks to use for each one, or ask students to pick a suitable scenario where each of the different types of validation checks could be used.
- Give students code that performs some validation and a list of values and get them to identify which values would be accepted and which would be rejected by the code.
- Give students code that is supposed to carry some validation but that contains errors and ask them to correct it. This could be done on screen or on paper. Errors might focus on incorrect conditions (such as mixing up <, <= and > or AND and OR) or other aspects of the code.
- Get students to write code to perform the different types of validation checks.
- Get students to add code to perform the different types of validation checks to programs that they have already written.
- Give students working code that performs a validation check using a pre-conditioned or post-conditioned loop and get them to convert it to use the other type of loop.
- Get students to write code to perform validation using subroutines and parameters.

Authentication routines

The use of usernames and their associated passwords is a very common method of authentication. For a single username and password, or a small number of these, authentication could be carried out using direct comparisons to the allowable values. If there are many possible usernames and passwords then it would make sense to use a data structure such as an array or list to store these.

The following examples show how authentication can be achieved for a single username and password and for a list/array of usernames and passwords in C#, Python and VB.NET.

C# single username and password

```
string username;
string password;
do
{
  Console.Write("Enter username:");
  username = Console.ReadLine();
  Console.Write("Enter password:");
  password = Console.ReadLine();
  if (username != "US001" || password != "ABC123")
  {
    Console.WriteLine("Invalid login");
  }
} while (username != "US001" || password != "ABC123");
Console.WriteLine("Logged in");
```

C# array of usernames and passwords

```
string[] validUsernames = { "US001", "US002", "US003" };
string[] validPasswords = { "ABC123", "DEF456", "GHI789" };
string username;
string password;
bool loggedIn = false;
Console.Write("Enter username:");
username = Console.ReadLine();
Console.Write("Enter password:");
password = Console.ReadLine();
for (int i = 0; i < validUsernames.Length; i++)
{
  if (validUsernames[i] == username && validPasswords[i] ==
password)
  {
    loggedIn = true;
    Console.WriteLine("Logged in");
  }
}
if (loggedIn == false) Console.WriteLine("Invalid login");
```

Python single username and password

```
username = input("Enter username:")
password = input("Enter password:")
while username != "US001" or password !="ABC123":
  print("Invalid login")
  username = input("Enter username:")
  password = input("Enter password:")
print("Logged in")
```

Python array of usernames and passwords

```
validusernames = [ "US001", "US002", "US003" ]
validpasswords = [ "ABC123", "DEF456", "GHI789" ]
loggedin = False
username = input("Enter username:")
password = input("Enter password:")
for i in range(len(validusernames)):
  if validusernames[i] == username and validpasswords[i] ==
password:
    loggedin = True
    print("Logged in")
if (loggedin == False):
  print("Invalid login")
```

VB.NET single username and password

```
Dim username As String = ""
Dim password As String = ""
While username <> "US001" Or password <> "ABC123"
  Console.Write("Enter username:")
  username = Console.ReadLine()
  Console.Write("Enter password:")
  password = Console.ReadLine()
  If username <> "US001" Or password <> "ABC123" Then
    Console.WriteLine("Invalid login")
  Else
    Console.WriteLine("Logged in")
  End If
End While
```

VB.NET array of usernames and passwords

```
Dim validUsernames As String() = {"User001", "User002", "User003"}
Dim validPasswords As String() = {"ABC123", "DEF456", "GHI789"}
Dim username As String
Dim password As String
Console.Write("Enter username:")
username = Console.ReadLine()
Console.Write("Enter password:")
password = Console.ReadLine()
Dim i As Integer = -1
Do
  i = i + 1
Loop Until validUsernames(i) = username And validPasswords(i) =
password Or i = validUsernames.Length - 1
If validUsernames(i) = username And validPasswords(i) = password
Then
  Console.WriteLine("Logged in")
Else
  Console.WriteLine("Invalid login")
End If
```

Suggested student activities

- Get students to write code that accepts a single username and password. Initially this could display a valid/invalid login message then it could be extended to repeat until a valid set of credentials are entered.
- Get students to write code that stores multiple usernames and passwords in a data structure and then accepts or rejects a set of credentials entered by the user. This could be achieved using two arrays or lists or an array or list of records, where each record stores both a username and password.
- Give students code for an authentication routine that has multiple usernames and passwords and get them to complete a trace table for a particular set of input values.
- Get students to add a login system to an existing program that they have written. Once working, this could be extended to include the concept of different users having different rights or to unlocking hidden features.
- As an extension task, this topic could be used to illustrate code efficiency to students, for example:
    - Different versions of code that used a linear search that terminated as soon as a matching username and password in an array/list was found or that only terminated when the end of an array/list was reached could be compared.
    - The use of a built-in function (such as Array.IndexOf in C# or VB.NET or index in Python) to search through an array or list could be compared to a student writing their own code to achieve the same purpose.