TEMASEK
JUNIOR COLLEGE

Temasek Junior College
JC H2 Computing
Problem Solving & Algorithm Design 10
Data Capturing and Correctness

1 Data Validation

Example 1 CW2011(A) – use for discussion

The examinations department of a school needs to store data on the examinations taken by its students. Examination subjects are classified as one of two types.

A subject can be classified as academic. The assessment process for an academic subject comprises two written papers. The duration of each of these two papers will be stored.

A subject can be classified as practical. For the practical assessment process there are no written papers but there is a single practical examination. A practical examination has a duration and a deadline date. The duration is the maximum time allowed for students to complete the work to be assessed. The deadline date is the date by which the practical assessment must be completed.

The examinations department decides to store the following data:

- **SubjectCode** is used to uniquely identify a particular subject and is four digits.
- **Name** is the name of the subject and is at most 30 characters.
- **SubType** is the type of subject and can have the values 'A' or 'P'.
- **P1Len** is the duration of the first written paper and can have the values 2.0, 2.5 and 3.0. For a practical subject this field would have the value 0.0.
- **P2Len** is the duration of the second written paper and can have the values 2.0, 2.5 and 3.0. For a practical subject this field would have the value 0.0.
- **PracLen** is the duration of the practical examination and can have values in the range 10.0 to 40.0. For an academic subject this field would contain the value 0.0.
- **CDate** is the final date by which the practical examination should be completed. It is six characters and is in the form 110504 (which represents 4th May 2011). For an academic subject this field would have the value '000000'.

A module, **ADDSUBJECT,** which when called will allow the user to enter data on a new subject.

This data is then written to a text file named **EXAMS.DAT.**

EXAMS.DAT has the following structure:

```
<NoOfRecords><UpdateDate>
<SubjectCode><Name><SubType><P1Len><P2Len><PracLen><CDate>
<SubjectCode><Name><SubType><P1Len><P2Len><PracLen><CDate>
```

NoOfRecords is the number of records stored in file
UpdateDate is the date that the file was last updated and is in the form DD-MM-YYYY.

Discuss with details the **validations** need to apply for the data to be **captured** before they **store** into the file EXAM.DAT.

Draw flowcharts for the validations of DD-MM-YYYY, SubjectCode, P1Len and CDate.

EXAMS.DAT Sample data:

```
10|05-05-2021
5566|Math|A 2.5 3.0 0.0 000000
6846|Physic|A 2.0 3.0 0.0 000000
5846|Computing|P 0.0 0.0 15.0 110430
0023|Biology|A 2.5 2.0 0.0 000000
1234|Chemistry|A 2.0 3.0 0.0 000000
3258|China Studies|A 3.0 3.0 0.0 000000
7789|Literature|A 2.5 3.5 0.0 000000
9999|Geography|A 2.5 2.0 0.0 000000
```

1.1 Why Validation is needed?

For a problem to be solved, its inputs, outputs and processes need to be defined clearly. While the programmer often has control over the processes used and outputs produced by the code, the supplied inputs can come from many possible sources, and the programmer often has no control over what is supplied. This means that the supplied inputs may not actually meet the requirements for valid or acceptable input data as defined by the problem.

e.g Data to be captured is an numeric value but supplied input is a string, etc.

```
>>> age = int(input ("Please enter you age:"))
Please enter you age:4o
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    age = int(input ("Please enter you age:"))
ValueError: invalid literal for int() with base 10: '4o'
```

Data validation prevents the program from behaving in an unexpected manner by performing data validation before the data is processed. This ensures that the input data is sensible, complete and within acceptable boundaries.

1.2 What should we do if data is invalid?

When performing data validation, there are two basic choices on what to do if the data is invalid.

(a) Try Again
For instance, if the data was entered by the user, we can ask for the data to be re-entered.
This option makes the most sense if the data might change by trying again.

(b) Quit
If the input data is invalid and was not entered by the user (i.e., it was read from a file), we can simply end the program early.
In Python, one way of doing this is by importing the sys (short for system) module and calling `sys.exit()`.

2 Common Data Validation Techniques

As data is entered into the Command Line Interface (CLI) console-based input or form of Graphic User Interface (GUI) application, it needs to be checked for **accuracy**.

Two techniques help us do this: **validation** and **verification**.

Command Line Interface	Graphic User Interface
`>>> age = int(input ("Please enter you age:"))` `Please enter you age:`	Please enter you age Seventeen

Validation is the **automatic** checking of data entered into a computer system and to check that the data is **sensible,** the captured data does not mean that the data is the **right data**.
[e.g. no dates of birth in the future **12-02-2025**, etc.];

- Validation involves using the properties of the data to identify any inputs that are obviously wrong.
  SubjectCode is used to uniquely identify a particular subject and is **four digits**.
  e.g. Input SubjectCode: 99**OI**

- Validation only proves that the data entered is a reasonable value for the computer to accept.
  e.g. Input Age of JC1 students: **23** // 15 to 21
  Input SubType: **'H'** // 'A' or 'P'

- It cannot prove that the data entered is the actual value the user intended.
  e.g. Input Height of JC1 student Tan Lee Ming: **1.66** //Actually is 1.56
  Input LastDate: **12-12-20** //31-11-2020

- However, it does allow the computer to filter out obvious mistakes
  e.g. Input Height of JC1 student Tan Lee Ming: **2.96** //No such value
  Input duration of the first written paper: **3.5** //2.0, 2.5 and 3.0

- The data is not processed **until** the validation succeeds
  [which program construct will be used?].

The checking is done by software that can either be part of the input system or a separate program that checks the data at the following stage
[Is this check by the computer automatically?].

Checks like this are called **validation checks**.

Different validation checks can be used on different fields, depending on the **type** of data being entered

2.1 Validation checks

2.1.1 Presence check

For some problems, portions of the input are required (or mandatory) while other parts may be optional. If it is possible to leave out any optional inputs, the program should perform a **presence check** to ensure that all the required inputs are provided.

When a presence check is set up on a field then that field cannot be left blank, some data must be entered into it.

To ensure that all necessary fields are present.

The presence check does not check that the data is correct, only that some data is present. This checks that an entry has been made for the field.

e.g. **Name** is the name of the subject cannot be left blank is a required field.

Pseudocode:

```
Valid_flag ← False
IF LENGTH(Name) <> 0 THEN   // Zero
     Valid_flag ← True
ENDIF
```

Name of subject
[at most 30 characters]

2.1.2 Type check

To ensure a data item is of a particular data type.

Practical examination
(10.0 to 40.0 minutes)

e.g. **PracLen** is the duration of the practical examination and can have values in the range 10.0 to 40.0.

Partial Python code: (Can use pseudocode for type check?)

```
Valid_flag = True
#Validation rule: Data type - float
if (type(float(PracLen))) != float:
   print (f"\tError! '{PracLen}' PracLen is not real!")
   Valid_flag ← False
```

☞ e.g. The age will be entered as a string (but represent a whole number).
- How to check an input as a string which represent an integer in Python?
- Can we use function `decimal()` or method `isdigit()`?

2.1.3 Existence check

Identifies whether a certain value is present in a specified area

e.g. **SubType** is the type of subject and can have the values 'A' or 'P'

Pseudocode:

```
Valid_flag ← True
//Validation rule:
IF NOT((SubType = 'A' OR SubType = 'P')) THEN
     PRINT ("Error! Subject Type can only be A or P.")
     Valid_flag ← False
ENDIF
```

Subject Type (A or P)

2.1.4 Length check

The length of the inputs for a problem often needs to meet certain requirements in order to be valid.

e.g. **Name** is the name of the subject and is at most 30 characters

Pseudocode:

```
Valid_flag ← True
IF LENGTH(Name) >  30 THEN  //or other skill
      Valid_flag ← False
ENDIF
```

Name of subject
[at most 30 characters]

2.1.5 Range check

To ensure that data value is within a pre-determined range.

This checks a value to be within a certain range of values.

Practical examination
(10.0 to 40.0 minutes)

e.g. PracLen is the duration of the practical examination and can have values in the range 10.0 to 40.0.

Pseudocode:

```
Valid_flag ← True
IF PracLen  < 10.0 OR PracLen > 40.0 THEN
      PRINT 'Duration of the practical examination is out of range. '
      Valid_flag ← False
ENDIF
```

2.1.6 Format check (also called a picture check)

To ensure a data item matches a previously determined pattern and that particular characters have particular values such as being letters or digits.

This checks that data is of the right format, that it is made up of the correct combination of alphabetic and numeric characters.

e.g. UpdateDate is the date that the file was last updated and is in the form DD-MM-YYYY

Date of updating (DD-MM-YYYY) _ _

Input UpdateDate as a string must check:

(a) Length (10 characters)
(b) Third and sixth characters are '–'
(c) The rest are digits
(d) Range of DD
(e) Range of MM
(f) Range of YYYY
(g) Leap year consideration

e.g. An Identification number (Singapore resident) must be in the form of X9999999X. The first and the last characters must be letters. The other seven characters are numbers. The total length is nine characters. Any other format is rejected.

Identification number[X9999999X]

To ensure the individual characters that make up the data are valid - e.g. no letters in numerical data.

e.g. Credit/Debit card is an 16-digits string '3014 5800 0013 3788'

Card Number*

2.1.7 Check Digit

Allows a number to be self-checking.

NRIC[X9999999X] T0208480A

This check is used with only large numbers where a great possibility to mistype numbers.

The last one or two digits on a code are used to check the other digits are correct and this is done by cutting off the last digit from the original number and perform a specific mathematical operation on the remaining numbers to get a new digit and compare it with the one originally have been cut off, if it does not match it means the original number has been mistyped.

2.1.8 Integrity checks

To confirm the value of a piece of data by comparing it with other data. E.g. an individual's date of birth can be compared with their current age, if known.

Date of birth[dd/mm/yyyy]

Age

2.1.9 *Lookup checks*

To ensure that the data matches one of a limited number of valid entries.

e.g. **P1Len** is the duration of the first written paper and can have the values 2.0, 2.5 and 3.0.

Duration of the first written paper (2.0, 2.5 and 3.0 hours)

Pseudocode:

```
Valid_flag ← False
IF P1Len = 2.0 OR P1Len = 2.5 OR P1Len = 3.0 THEN
     Valid_flag ← True
ENDIF
```

e.g. subjects studied in a school should be selected from a list of Mathematics, English etc.

A ***file lookup check*** *is* used to compare the data against a larger list of items that are held in a data file.
e.g. the product code of an item in a supermarket will be looked up in the stock file to confirm such an item exists.

2.1.10 *Batch header check*

This is concerned with Batch processing.

The total number of records in the batch should be calculated by the computer and compared with the figure on the batch header. The control totals and hash totals are also calculated and compared.

EXAMS.DAT Sample data:

```
10|05-05-2021
5566|Math|A 2.5 3.0 0.0 000000
6846|Physic|A 2.0 3.0 0.0 000000
5846|Computing|P 0.0 0.0 15.0 110430
0023|Biology|A 2.5 2.0 0.0 000000
1234|Chemistry|A 2.0 3.0 0.0 000000
3258|China Studies|A 3.0 3.0 0.0 000000
7789|Literature|A 2.5 3.5 0.0 000000
9999|Geography|A 2.5 2.0 0.0 000000
8463|Art|P 0.0 0.0 30.0 110829
2317|Chinese|A 2.0 2.0 0.0 000000
9517|Math|A 2.5 3.0 0.0 000000
```

2.2 Check Digit

This is used to check the validity of code numbers, for example product codes in a supermarket or bank account numbers. These numbers are long and prone to data entry errors. It is crucial that such numbers are entered correctly so that the right record in the file or database is identified.

A check digit is an extra digit added to the end of the original code. The value of the check digit is determined by the value and positioning of the other digits: for any given code there is only one possible check digit. When code has been entered, the check digit is calculated and compared to the entered value. If the two digits do not match, an error is reported.

Check digit calculator [https://www.gs1.org/services/check-digit-calculator]	
The last digit of a barcode number is a computer check digit which makes sure the barcode is correctly composed.	

- Also used to validate data captured by bar code reader.
- Validation is performed on input.

Note that check digits are used to validate numbers that are being used as identifiers rather than as numbers. There is no need to store the check digit within the computer once the input has been validated.

2.2.1 Check digit using Modulo 11

The check digit is the digit in the unit column (1). The actual data is the number from position 2 upwards.

To find the check digit to attach to a number perform the calculation as from position 2. The check digit is what must be added to the result to make it **divisible** by 11 [**i.e. the reminder of the sum of all products divided by 11 is 0-ZERO!**].

Example 2: Checking 83232 is valid with modulo 11

Position or Weight	5	4	3	2	1
Digit	8	3	2	3	2←check digit
Product of (Weight and Digit)	40	12	6	6	1
Sum of Products (S)	66				
Remainder of S/11	0 [S is exactly **divisible** by 11]				

This shows validation of the number 83232. Note that the final value 2 is the check digit. The number is valid because the final total is exactly **divisible** by 11.

Note that 10 is a possible value for the check digit. This is represented by an X. Use of a check digit is guaranteed to detect the two most common human errors in entering a long sequence of numbers - entering one digit incorrectly or transposing two digits. Used whenever a particular long number is to be used repeatedly - e.g. part number or account number.

Example 3 Find check digit of 1238 ␣

Given number without check digit	1	2	3	8	␣

Say:
I will adopt the method of modulo 11.

					Check Digit
Weight or Position	5	4	3	2	
Digit	1	2	3	8	
Weight x Digit	5	8	9	16	
Sum of Products (S)	**38**				
Remainder of S/11 (R)	**R** = (Reminder of 38/11) = 5				
Difference of 11 and R	11 – **R** = 11 – 5				6
Number with check digit	1	2	3	8	6

Validation of 12386:

Given number with check digit	1	2	3	8	6
Weight or Position	5	4	3	2	1 [*]
Digit	1	2	3	8	6
Position x Digit	5	8	9	16	6
Sum of Products (S)	44				
R =(Reminder of **S**/11) = 0	The number is valid because reminder is 0.				

5 4 3 2 1
8 3 3 3
40 12 9 6
67

67 ÷ 11 = 6 R1
11 − 1 = 10 check digit = X.

this weight

8	3	3	3	X
5	4	3	2	1
40	12	9	6	10
		77 ÷ 11 = R0		

Example 4 NRIC is used to identify a Singapore resident and is at exactly 9 characters. **[Reference]**

- The first and the last characters are upper case letter and the remaining seven are digits, e.g. S8524223G.
- Validation code for NRIC. For NRIC number in the format S1234567D, T1276249B and G2873148M
- Algorithm for generating check digit
  1. SUM = (2*A) + (7*B) + (6*C) + (5*D) + (4*E) + (3*F) + (2*G)
  2. Add 4 to SUM if your IC number starts with T or G.
  3. Find the remainder (RD) when SUM is divided by 11.
  4. Subtract remainder (RD) from 11 and get the difference (DIFF)
  5. Convert the difference (DIFF) into a letter using the following table.

	1	2	3	4	5	6	7	8	9	10	11
Letter (NRIC)	A	B	C	D	E	F	G	H	I	Z	J
Letter (FIN)	K	L	M	N	P	Q	R	T	U	W	X

NRIC	S	8	5	2	4	2	2	3	G
Numeric digit [D]		8	5	2	4	2	2	3	
Weight [W]		2	7	6	5	4	3	2	
PRODUCT [D x W]		16	35	12	20	8	6	6	
SUM(PRODUCT) =103	RD = SUM(PRODUCT) % 11 = 4 % - reminder operator					DIFF = 11 – RD = 11 – 4 = 7			Check Character = G

NRIC	T	0	2	3	6	2	8	3	F
Numeric digit [D]		0	2	3	6	2	8	3	
Weight [W]		2	7	6	5	4	3	2	
PRODUCT [D x W]		0	14	18	30	8	24	6	
SUM(PRODUCT) = 100 SUM = SUM + 4 =104	RD = SUM(PRODUCT) % 11 = 5 % - reminder operator					DIFF = 11 – RD = 11 – 5 = 6			Check Character = F

NRIC	G	1	6	0	4	5	3	0	L
Numeric digit [D]		1	6	0	4	5	3	0	
Weight [W]		2	7	6	5	4	3	2	
PRODUCT [D x W]		2	42	0	20	20	9	0	
SUM(PRODUCT) =93 SUM = SUM + 4 = 97	RD = SUM(PRODUCT) % 11 = 9 % - reminder operator					DIFF = 11 – RD = 11 – 9 = 2			Check Character = L

Test data: S8524223G, S8772761J, S8703660Z, S1234567D, T1276249B and G2873148M.

- Validation of NRIC: Given S9975149E T0208480A, G0769459N

NRIC	S	9	9	7	5	1	4	9	E [5]
Numeric digit [D]		9	9	7	5	1	4	9	5
Weight [W]		2	7	6	5	4	3	2	1
PRODUCT [D x W]		18	63	42	25	4	12	18	5
SUM(PRODUCT) =187		RD = SUM(PRODUCT) % 11 = 187 % 11 = 0				S9975149E is **valid because reminder is 0**.			

NRIC	T	0	2	0	8	4	8	0	A [1]
Numeric digit [D]		0	2	0	8	4	8	0	1
Weight [W]		2	7	6	5	4	3	2	1
PRODUCT [D x W]		0	14	0	40	16	24	0	1
SUM(PRODUCT) = 95 SUM = SUM + 4 = 99			RD = SUM(PRODUCT) % 11 = 99 % 11 = 0				T0208480A is **valid because reminder is 0.**		

Tutorial 10.1 [Validation] Q1, 2, and 4

Mini Project
Complete all validation checks of Example 1.

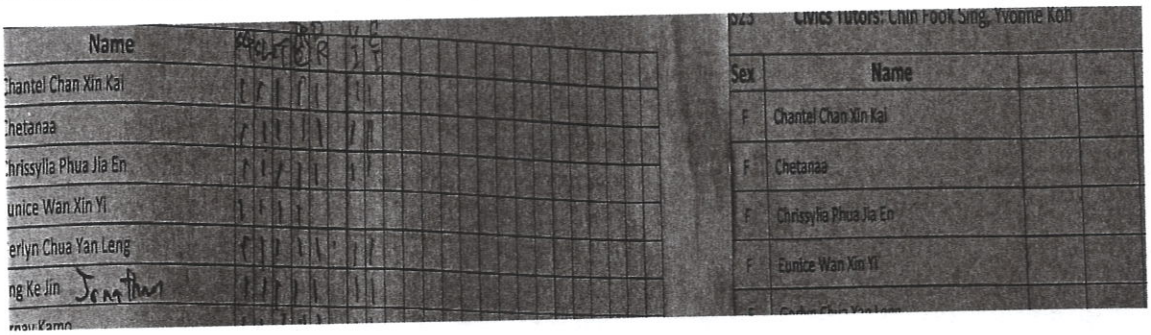
Given:

```
DECLARE NoOfRecords : INTEGER
DECLARE SubjectCode : STRING
DECLARE Name : STRING
DECLARE SubType : CHAR
DECLARE P1Len : REAL
DECLARE P2Len : REAL
DECLARE PracLen : REAL
DECLARE CDate : STRING

// Validation of NoOfRecords
REPEAT
      Valid_Flag ← True
      PROMPT "No. of records[1 - 30]: "
      INPUT NoOfRecords
      IF TYPE OF NoOfRecords is not INTEGER THEN      //type check
            PRINT "Error! No. of records can only be an integer [1-30]."
            Valid_Flag ← False
      ELSEIF NoOfRecords < 1 OR NoOfRecords > 30 THEN //Range check
            PRINT "Error! No. of records can only be in range [1-30]."
            Valid_Flag ← False
      ENDIF
UNTIL Valid_Flag = True

//LOOP NoOfRecords times
DOWHILE NoOfRecords > 0
      REPEAT // Validation of SubjectCode
            Valid_Flag ← True
            PROMPT "Subject Code: "
            INPUT SubjectCode
            IF LENGTH(SubjectCode) <> 4 THEN //Presence and length check
                  PRINT "Error! Subject Code must be 4 digits. "
                  Valid_Flag ← False
            ELSEIF NOT every character is a digit THEN      //Format check
                  PRINT "Error! Subject Code contains only digits only. "
                  Valid_Flag ← False
            ENDIF
       UNTIL Valid_Flag = True

       REPEAT // Validation of Name
            Valid_Flag ← True
            PROMPT "Name: "
            INPUT Name
            NameLength ← LENGTH(Name)
            //Presence and length check
            IF NameLength < 4 OR NameLength > 30 THEN
                  PRINT "Error! Name must be at most 30 characters. "
                  Valid_Flag ← False
            ENDIF
       UNTIL Valid_Flag = True
```

```
      REPEAT // Validation of SubType
            Valid_Flag ← True
            [Fill in pseudocode for presence and existence checks]




      UNTIL Valid_Flag = True

      //Handling different papers
      IF SubType = 'A' THEN          //Written papers
            PracLen ← 0.0            //Why?
            REPEAT // Validation of P1Len
                  Valid_Flag ← True
                  [Fill in pseudocode for type and look up checks]



            UNTIL Valid_Flag = True

            REPEAT // Validation of P2Len
                  Valid_Flag ← True
                  [Fill in pseudocode for type and look up checks]



            UNTIL Valid_Flag = True
      ELSEIF SubType = 'P' THEN      //Practical subjects
            P1Len ← 0.0              //Why?
            P2Len ← 0.0              //Why?
            REPEAT // Validation of PracLen
                  Valid_Flag ← True
                  [Fill in pseudocode for type and range checks]



            UNTIL Valid_Flag = True

      REPEAT // Validation of CDate
            Valid_Flag ← True
            [Fill in pseudocode for presence and format checks]



      UNTIL Valid_Flag = True
      NoOfRecords ← NoOfRecords - 1        //Why?
ENDDO
```

3 Verification

When data is transcribed from one medium to another there is always a danger that errors will be introduced. The copy will not then be the same as the original. This is particularly true if the transcription is being done manually as when a human operator reads a source document and uses a keyboard to transcribe the data to a computer readable format (key to disc systems) and when it is copied between other components within *a* computerised system (including from remote sensors or other computer).



Verification is used to check that data is entered correctly and that there are no transcription errors.

If data has been copied automatically from one format to another then the computer will automatically compare the two versions and inform the user if there are any differences.

If a keyboard operator has done the transcription then this type of check is impossible. In this situation the data is re-entered by a different operator with the computer system checking for differences as the second copy is entered. Any differences must be the result of a transcription error by one of the two operators. In this case the transcribed versions can be checked manually against the source document and the error corrected.

You have probably come across another example of verification when setting a new password

e.g. you are usually asked to key the password in a second time to ensure that you didn't make a keying error the first time, as it is not echoed on the screen.

The main aim of verification is to trap transcription errors – errors in transferring the source data into the computer.

It cannot guarantee accuracy:

- If the original data was incorrect the data will be entered wrongly
- If the data was entered twice, it may have been entered twice incorrectly
- If the data was being manually checked, mistakes can creep in.

3.1 Transcription Error

A **transcription error** is a specific type of data entry error that is commonly made by human operators or by optical character recognition programs (OCR).

Human transcription errors are commonly the result of typographical mistakes, putting fingers in the wrong place during touch typing is the easiest way to ascertain this error.

Examples of Transcription Error
Input : Gtegory
Instead of : Gregory
Input : 34rd of August
Instead of : 23rd of August
Input : Jishua
Instead of : Joshua

Electronic transcription occurs when the results of a scan of printed matter is compromised or in an unusual font e.g. - if the paper is crumpled, or the ink is smudged when wet, the OCR may have a transcription error when reading.

3.2 Transposition Error

Transposition errors are commonly mistaken for transcription errors, but they should not be confused. As the name suggest, transposition errors occur when characters have "transposed" — that is, they have switched places.

Transposition errors are almost always human in origin.

Examples of Transposition Error
Input : Gergory
Instead of : Gregory
Input : 23rd of Auguts
Instead of : 23rd of August
Input : Johsua
Instead of : Joshua

The most common way for characters to be transposed is when a user is touch typing at a speed that makes them input one character, before the other.

3.3 Solving Transcription and Transposition Errors

The most obvious cure for the errors is for the user to **watch** the screen when they type, and to **proof read**. If the entry is occurring in data capture forms, databases or subscription forms, the coder of the forms or the database administrator should use input masks or validation rules.

In computer programming, an **input mask** refers to a string expression, defined by a developer that governs what a user is allowed to enter in as input in a text box. It can be said to be a template, or set format that entered data must conform to, mainly used for the purposes of data integrity by preventing transcription errors. The syntax of this string expression differs from implementation to implementation, but the fundamental input types are all supported.

Some frequent uses of input masks include entry of telephone numbers, credit card number, postal codes, times and dates.

e.g. When entering into a text box a phone number on a data capture form, in the format "(65)65646878" the area code brackets, the space between the number and the area code will automatically be placed in.

Generally speaking, an input mask is a user generated set of rules, so if say: only allows 45 characters, then the entry must stay up to or within 45 characters, they cannot exceed this rules. This kind of string is useful in finding reports and healthcare files.

Transcription and Transposition error may also occur in syntax when coding or programming, within variable declaration or within coding parameters - this should be checked by proof reading, however syntax errors may be picked up by the program the author is using.

Common desktop publishing and word processing applications use spell checkers and grammar checkers which may pick up on some transcription/transposition errors - not all errors may be picked up however, as some errors may form new words which fit grammatically.

For instance if the user wished to write "The fog was dense", but instead put "The dog was dense", a grammar and spell checker would not notify the user because both phrases are grammatically correct, as is the spelling of the word "dog" and "fog".

Unfortunately, the situation regarding these errors is likely to get worse before it gets better, as workload for users and workers regarding manual direct data entry devices (DDE) increases.

Double entry may also be leveraged to minimize transcription or transposition error, but at the cost of a reduced number of entries per unit time.

Summary The verification checks at the human-machine interface may include:

- ***Double entry verification*** to ensure data typed into a computer system entered accurately. The data is entered twice, by different operators, and compared by the system. Any differences can be identified and manual corrected.
  The computer compares the two sets of data to see if they match. If not it generates an error and a person will need to correct the mistake.
  Double-entry takes more time and effort, but it catches almost every mistake.

- ***Screen verification (Proof Reading)*** to ensure data being entered into a computer system accurate. After being entered the data is displayed on the screen, the user reads it and confirms if it is correct.
  If mistakes are spotted they can be corrected by the person.
  Proof-reading is quick and simple, but doesn't catch every mistake.

- ***Check digits*** are extra digits added as part of a numeric data item (like a code or a stock number). They are worked out from the other digits in way that can be repeated, enabling the data to be checked at a later state by working out the check digit again and comparing the results.

- ***Batch totals*** are the total value of one or more fields in a batch of data. They are calculated in advance (manually) and then compared with the total as calculated by the computer.

- ***Control totals*** are batch totals that have a meaningful value, for exam the total value of a batch of orders or simply a count of the number transactions.

- ***Hash totals*** are batch totals that have no other meaning, for example the total formed by adding all the dates of the orders or the total of all the numeric fields for a particular record.

Tutorial 10.2 [Verification] Q1, 2 and 3

Tutorial 10.1 [Validation]

1. Validation is a way of checking data in a database. [ ]

Name the most appropriate validation check that matches each description below. [4]

(a) Makes sure that the data entered into a field is exactly 10 characters
(b) Makes sure that a number lies between 10 and 100
(c) Makes sure that the data entered is numeric
(d) Makes sure that the date is entered as DD/MM/YYYY

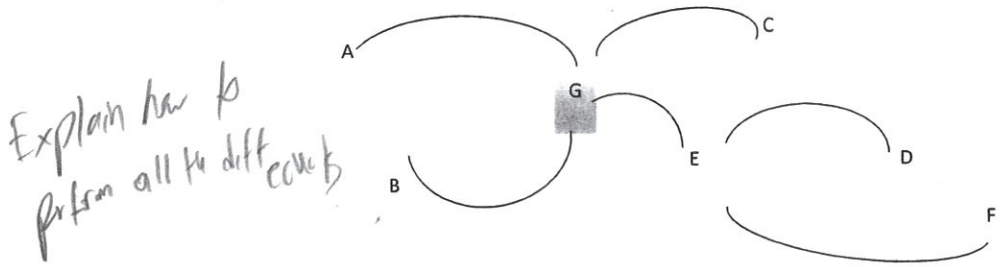
2. An airline keeps a database of all the flights it operates. A short extract is shown below. [ ]

Flight_number	Departure_Airport_Code	Number_of_passengers	Destination_Airport_code	Ticket_prices
EK236	JNB	135	LHR	$1200
EK080	DXB	256	DEL	$500
EK029	LHR	375	DXB	$650

(a) Explain fully, what is meant by the following validation checks using the extract above.
Range check/ Format check/ Length check [6]

(b) Using the extract above, give an example of **one** field which would contain:
Text data (alphanumeric)
Numeric data [2]

3. A train company runs a rail system that serves a number of small towns by connecting them to a major city (G on the diagram). [ ]



Explain how to perform all the diff checks

At present the information system in the station at G comprises printed timetables on boards on each platform and a series of announcements if the timetable is to be altered.
It is decided to implement a system of terminals on each platform which are all connected to a central file server. Passengers will be able to make enquiries at the terminals about the times of trains.

In addition to the terminals, it is decided to place monitor screens above each platform. The monitors will supply information about trains due to leave the station.

The details of the trains are stored over a period of a week. The number of trains that are expected to leave the station in a week varies depending on the time of year, but is never more than 1000 a week. Trains are identified by a 5-digit code number. The first digit, between 0 and 5, identifies the station that the train is going to. The second digit, between 0 and 4, identifies the type of train, and the third and fourth digits provide an identifier of the specific train. The fifth digit is a check digit.

(a) The check digit acts as a validation check when the code is input to the system by an operator. Explain how this check digit may be calculated. [5]

(b) The train information is **stored** as follows.
(i) The code number of the train is stored as an integer.
(ii) The destination of the train is stored using ASCII characters.
(iii) Whether or not the train has refreshments available is stored as Boolean data.
Explain how each of these data types is stored and used. [6]

4. All books have a unique 10-digit number to identify them called the International Standard Book Number (ISBN). Each ISBN has four parts separated by hyphens: the Group identifier, the Publisher identifier, the Title identifier and the check digit. [GCEA 9349/05]

   - the Group identifier can be 2 or 3 digits;
   - the Group identifier and the Publisher identifier have together to be 5 digits;
   - the Title identifier is 4 digits;
   - the check digit is a single digit.

   The check digit is added to the ISBN after being calculated. Examples of ISBNs before the check digit is added are: 024-12-1479 and 02-435-3432

   (a) Describe an algorithm [flowchart] for the calculation of the check digit. Assume that the calculation uses modulus 11. [5]

   (b) Explain how the use of check digit helps to reduce the possibility of the incorrect entry of an ISBN when details of a new book are being entered. [2]

Tutorial 10.2 [Verification]

1. Tick whether the following statements apply to verification, validation or neither. [4]

[ ]

	Verification	**Validation**	**Neither verification nor validation**
Data is entered by two different operators			
Data is checked to see if it is present			
Data is checked to make sure it is correct			
Data entered is checked to see if it matches data on the source document			

2. There are two types of verification, visual and double data entry. [ ] [2]
   Explain the differences between visual verification and double data entry.

3. An examination board decides that its examiners who have internet access will enter examination marks on-line rather than using the existing manual methods for data capture. ( )

   (a) Examiners who do **not** have access to the internet must enter their marks on a form. State and justify **two** input methods that could be used. [4]

   (b) Explain the difference between data verification and data validation. [4]

   (c) (i) Describe a verification method which can be used when marks are entered by those examiners with internet access. [3]

   (ii) Describe a verification method that can be used by examiners who do **not** have internet access. [2]

   (d) State **two** validation checks that could be used in this application. For each check explain why it is appropriate. [4]