

Table of Contents

Topic 8: Databases & Microsoft Access 2016	3
Objectives	4
What Is A Database?	5
Data Versus Information	5
Data	5
Information	5
Organising Data	5
Tables	6
Records	6
Fields	6
Primary Keys	6
<i>Exercise 1 - Tables</i>	7
Data Types	8
Microsoft Access	9
Creating a Blank Database	9
Creating a Table	10
The Table Window	10
Adding Fields	11
Naming Fields	11
Creating a Primary key	12
Choosing Data Types	12
Describing Fields	13
Field Properties in Access	13
Adding Records	13
<i>Exercise 2 - Data Types</i>	14
Relational Databases	16
Relationships Between Tables	16
ONE-TO-ONE Relationships	17
ONE-TO-MANY Relationships	17
MANY-TO-MANY Relationships	17
The Problem with Many to Many Relationships	18
Transaction Tables	18
Database Management Systems (DBMS)	19
Advantages of DBMS	19
<i>Exercise 3 – Creating a Relationship</i>	20
Enforce Referential Integrity	22
Queries	23
Criteria	23
Creating a Query	24
Creating a Statistical Query	25
Performing an Average:	25
Finding the Number of Occurrences:	25
<i>Exercise 4 – Creating Queries</i>	26
<i>Exercise 5 – More Queries</i>	30
<i>Exercise 6 – Written Exercises</i>	32
<i>Exercise 7 – Field Properties</i>	34
Database Normalisation	37
Normalising an Example Table	37
First Normal Form (1NF):	37
Second Normal Form (2NF):	38
Third Normal Form: Eliminate Data Not Dependent On Key	38
<i>Exercise 8 – Database Normalisation</i>	39
Forms	45
Creating a Simple Form	45
Creating a More Complex Form	45
<i>Exercise 9 - Forms</i>	45
Reports	49
Parts of a Report	49
Creating a Report Using the Report Wizard	50
<i>Exercise 10 - Reports</i>	51
Review Exercise 1: Company	53
Review Exercise 2: Media Loft	55
Systems Development Life Cycle	57
Problem Definition	57

Analysis	58
Design	59
Development and Validation	60
Evaluation	61
Presentation of the Package	61
Marks Scheme For Individual Project	62
Definitions	65
Topic 9: Programming Using Javascript	66
Objectives	67
Computer Programs	68
Algorithms	68
Pseudo-Code	68
Control Structures	69
Deskchecks	69
Sequence Control Structure	70
<i>Exercise 1 – Sequence</i>	<i>72</i>
Selection Control Structure	73
Conditional Operators	73
<i>Exercise 2 – Selection</i>	<i>78</i>
Repetition Control Structure (Loops/Iteration)	80
Types of Loops	80
<i>Exercise 3 – Repetition</i>	<i>87</i>
Data Validation	89
More Than One Test Condition:	90
Basic Computer Processing Operations	91
<i>Exercise 4 – Algorithms</i>	<i>93</i>
Introduction to Javascript	94
Using the SCRIPT Tag	94
Embedding JavaScript in HTML	94
Specifying the JavaScript Version	94
Output – The Alert Function	94
<i>Exercise 5 – Programming</i>	<i>95</i>
Variables	96
<i>Exercise 6 – Variables</i>	<i>96</i>
<i>Exercise 7 – Assigning Values to Variables</i>	<i>97</i>
Data Types	98
<i>Exercise 8 – Types of Variables</i>	<i>98</i>
Input - The Prompt box	99
<i>Exercise 9 – Input</i>	<i>99</i>
Performing Calculations	100
Arithmetical Operators	100
<i>Exercise 10 – Performing Calculations</i>	<i>100</i>
Chaining	101
Making Statements	102
Comparison Operators:	102
Conditional IF	102
<i>Exercise 11 – Conditional Statements</i>	<i>102</i>
Using Comments	104
If – Else Statements	104
<i>Exercise 12 – Using Comments</i>	<i>104</i>
Loops in Javascript	106
The Pre-Test Loop	106
The Post-Test Loop	106
More Than One Test Condition:	107
<i>Exercise 13 – Using the WHILE Loop</i>	<i>108</i>
<i>Exercise 14 – JavaScript Practice Exercises</i>	<i>108</i>
Defining and Calling Functions	109
<i>Exercise 15 – Functions</i>	<i>109</i>
Function Parameters and Arguments	111
<i>Exercise 16 – Function Parameters & Arguments</i>	<i>111</i>
Forms	114
FORM Tag	114
INPUT Tag	114
<i>Exercise 17 – Input Tag</i>	<i>115</i>
Radio Buttons	115
<i>Exercise 18 – Radio Buttons</i>	<i>115</i>

Checkboxes	116
<i>Exercise 19 - Checkboxes</i>	116
Text Areas	117
<i>Exercise 20 - Textareas</i>	117
Buttons	118
<i>Exercise 21 - Buttons</i>	118
<i>Exercise 22 – More JavaScript Problems</i>	119
<i>Exercise 23 – JavaScript Practice Exercises Using Objects</i>	119
“For” Loops	120
JavaScript for Loop Example	120
<i>Exercise 24 – “For” Loop</i>	120
Getting Values Out Of Radio Buttons.....	121
<i>Exercise 25 – Getting Values Out of a Radio Button</i>	121
Elements of JavaScript	122

Foundation Studies (CIS)

Topic 8: Databases & Microsoft Access 2016



Objectives

1. Define a database
2. List the components of a database
3. Describe the major database structures
4. Describe the major ways of controlling data integrity
5. Describe the three different types of relationships
6. Create a relational database using Microsoft Access
7. Normalise a database to minimise data redundancy.
8. Select appropriate primary keys for tables
9. Select appropriate data types, field sizes, default values and validation rules for data in tables
10. Create Access queries using: More than one criteria and Boolean AND, OR and NOT operators, Complex criteria, Calculation, Statistics and grouping
11. Create Access reports using multiple grouping and including formulas and totals.
12. Create Access forms based: One table, Multiple related tables (sub forms)
13. Use the SDLC to create a database system for a company
14. Describe problems with a system
15. Find useful outcomes for a new system
16. Document the proof that a system meets its objectives

What Is a Database?

A **database** is an organised collection of data related to a specific topic. It allows you to retrieve, sort, and summarise information related to the data.

Data Versus Information

Data

Data is raw, unorganised facts that need to be processed. E.g. Each student's test score is data

Information

When data is processed, organised, structured or presented in a given context so as to make it useful, it is called Information. E.g. The class' average score is the information that can be determined from the given data

A database uses **data** to produce meaningful information to a user such as:

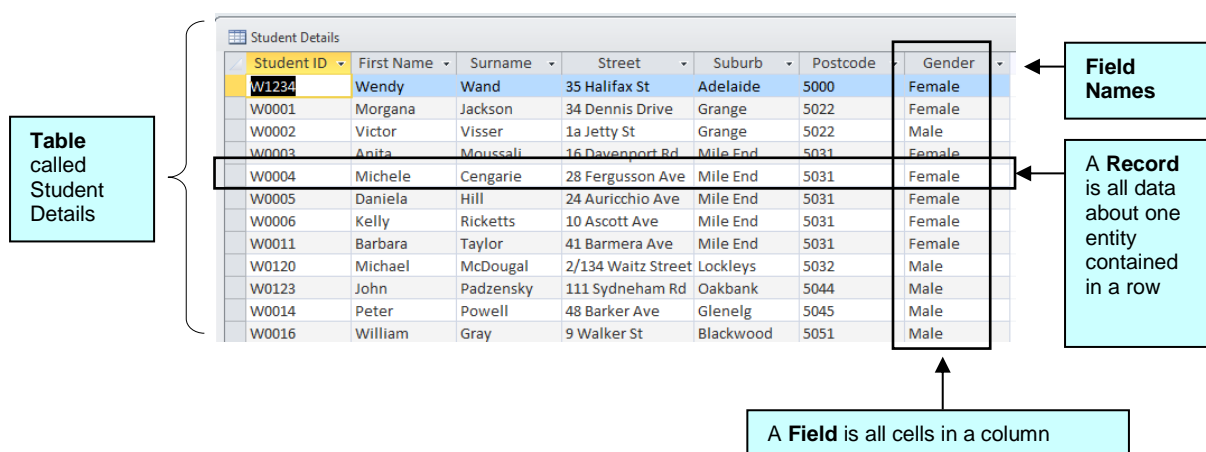
- Tomorrow's appointments in a hairdresser salon for an individual hairdresser;
- The rooms booked for next weekend in a motel;
- The DVDs available for rental;

Information can be used to make decisions such as:

- How often a movie will be screened in a cinema complex?
- How many staff will be needed at different times in a DVD store?
- On which flights special deals will be offered and how many seats will be released?
- How many cleaners will be required on different days in a motel?

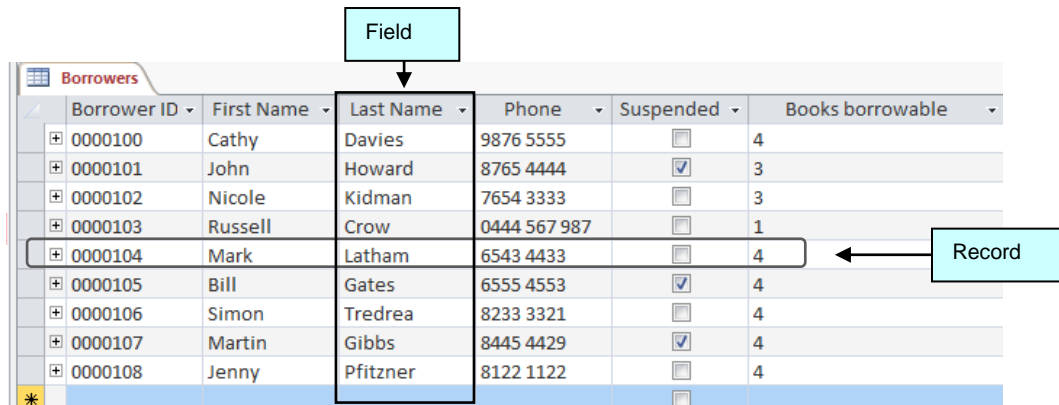
Organising Data

Database	a collection of related tables
Table (or File)	a collection of records that reflect a common meaning
Record	a related group of fields about an entity in a database
Field	a related group of characters



Tables

A **table** is a container for data about a particular subject, such as students. A table consists of records and fields. Each record (row) in a table contains information about an individual thing, such as a specific student. The records in a table are made up of fields (columns), such as names and phone numbers.



Borrower ID	First Name	Last Name	Phone	Suspended	Books borrowable
0000100	Cathy	Davies	9876 5555	<input type="checkbox"/>	4
0000101	John	Howard	8765 4444	<input checked="" type="checkbox"/>	3
0000102	Nicole	Kidman	7654 3333	<input type="checkbox"/>	3
0000103	Russell	Crow	0444 567 987	<input type="checkbox"/>	1
0000104	Mark	Latham	6543 4433	<input type="checkbox"/>	4
0000105	Bill	Gates	6555 4553	<input checked="" type="checkbox"/>	4
0000106	Simon	Tredrea	8233 3321	<input type="checkbox"/>	4
0000107	Martin	Gibbs	8445 4429	<input checked="" type="checkbox"/>	4
0000108	Jenny	Pfizer	8122 1122	<input type="checkbox"/>	4

Records

A record is all of the data or information about one person or one thing. In the example above, all the details about the borrower Mark Latham is a record.

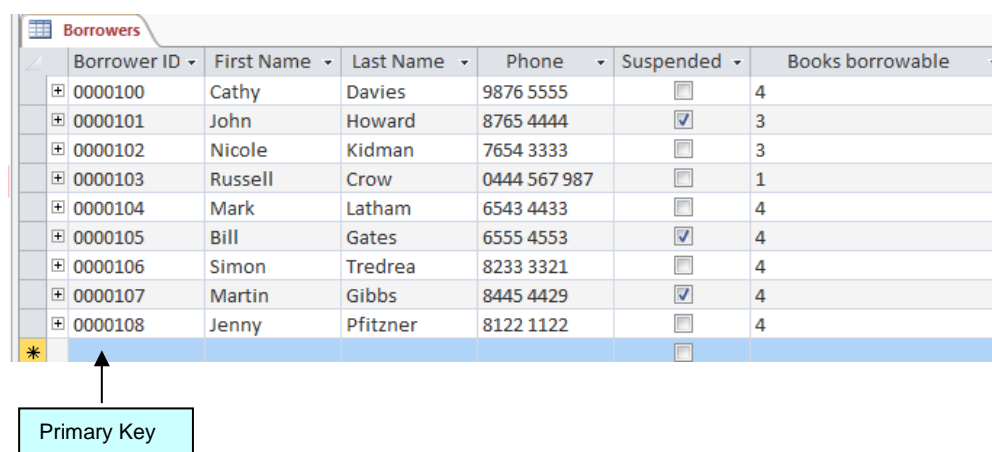
Fields

The records in a table can contain several categories of information. You add a field to your table for each category of information you want to store. For example, a table that stores records about books might have fields that store a book's number, the book's title and author.

Primary Keys

When a table is created, one of the fields is typically assigned as the **primary key**.

- A **primary key field** uniquely identifies each record
- Every record must have a different value for that field
- It prevents duplicates
- In the example below, Borrower ID in the example above would be classed as a primary key. Each value in the field is **unique**.



Borrower ID	First Name	Last Name	Phone	Suspended	Books borrowable
0000100	Cathy	Davies	9876 5555	<input type="checkbox"/>	4
0000101	John	Howard	8765 4444	<input checked="" type="checkbox"/>	3
0000102	Nicole	Kidman	7654 3333	<input type="checkbox"/>	3
0000103	Russell	Crow	0444 567 987	<input type="checkbox"/>	1
0000104	Mark	Latham	6543 4433	<input type="checkbox"/>	4
0000105	Bill	Gates	6555 4553	<input checked="" type="checkbox"/>	4
0000106	Simon	Tredrea	8233 3321	<input type="checkbox"/>	4
0000107	Martin	Gibbs	8445 4429	<input checked="" type="checkbox"/>	4
0000108	Jenny	Pfizer	8122 1122	<input type="checkbox"/>	4

Exercise 1 - Tables

Using the **Library.accdb** database, answer the following questions.

1. What is a table?

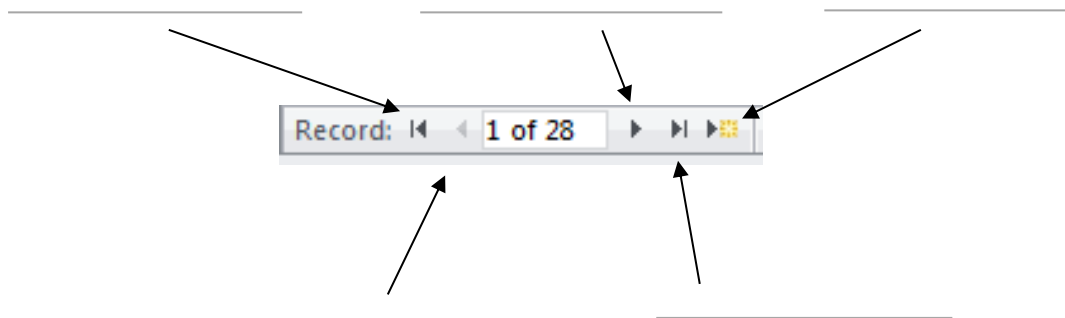
2. Write down all the fields used in the Borrowers table?

3. a. What is a record?

b. Write down the details of one record in the Borrowers table?

3. How many records are there in the Borrowers table? How did you find this out?

5. Describe what each of the buttons do in the diagram below.



Data Types

Access stores its data as fixed-length records. The size of each table is important, because a field that is too large will waste space, and too small will mean that the data cannot be accurately stored.

When you choose a field's data type, you're deciding:

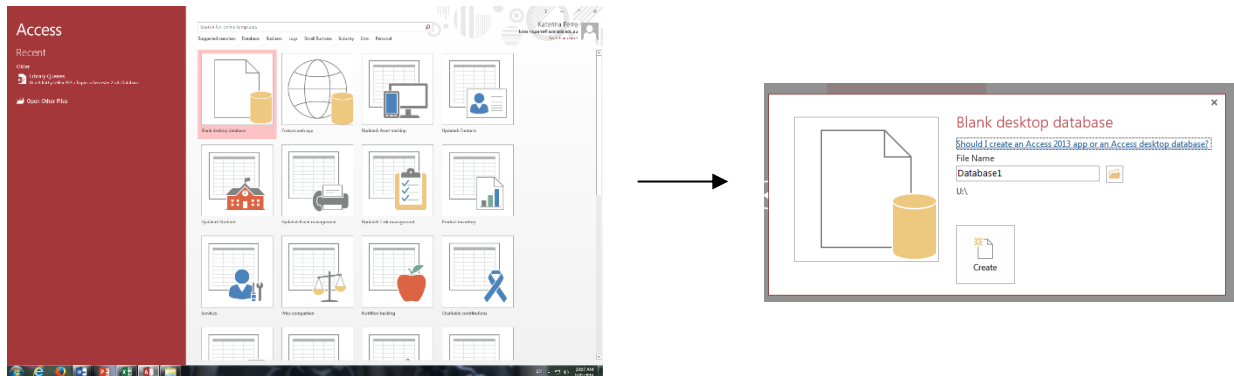
- What kind of values to allow in the field. For example, you cannot store text in a field with the **Number** data type.
- How much storage space Microsoft Access will set aside for the values stored in that field. For example, any value with the **Currency** data type uses 8 bytes of storage.
- What types of operations can be performed on the values in that field. For example, Access can find the sum of **Number** or **Currency** values but not of **Text**.

DATA TYPE	STORES	SIZE
Short Text	Alphanumeric text up to 255 characters	Up to 255 bytes (1 byte per character)
Long Text	A block of alphanumeric text that is more than 255 characters long and is formatted text.	Up to 32,000 bytes
Number	Numeric values that are not a monetary value and if the values in the field might be needed to perform a calculation.	
	Byte (1 byte)	e.g. 0 - 255
	Integer (2 bytes)	e.g. -32,768 – 32,767
	Long Integer (4 bytes)	E.g. -2,147,483,648 to 2,147,483,647.
	Single (4 bytes)	e.g. 3.7
	Double (8 bytes)	Large numbers with high level of precision
Date/Time	Dates and times	8 bytes
Currency	Monetary values	8 bytes
AutoNumber	A unique numeric value that Microsoft Access automatically increments for each record you add or randomly generates a number.	4 or 16 bytes (depends on Field Size property)
Yes/No	Boolean values	1 byte
OLE Object	OLE Objects, graphics, or other binary data e.g. photo from .bmp file	Up to 1 gigabyte (limited by disk space)
Hyperlink	Store web page or email addresses	The three parts of a hyperlink can contain up to 2048 characters
Attachment	Allows attaching files or images to a record. E.g. attach a photo of the contact, or attach documents such as a resume. For some file types, Access compresses each attachment as you add it.	
Calculated	A field that is based on a calculation of other fields in the same table	
Lookup Wizard	Starts the Lookup Wizard which creates a lookup field	4 bytes

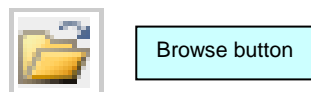
Microsoft Access

Creating a Blank Database

1. Open **Microsoft Access**.
2. You will be presented with a choice of templates to choose from. Select **Blank desktop database**.

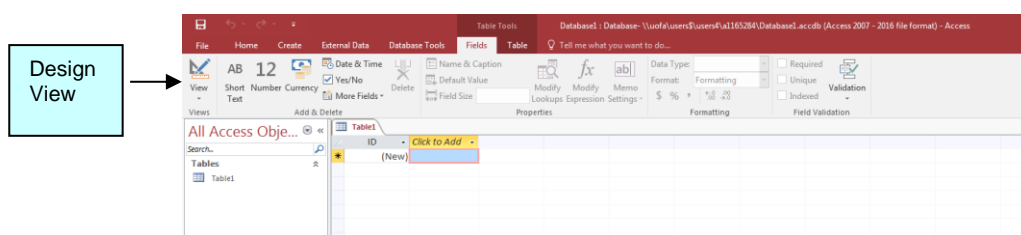


3. A **Blank desktop database dialog** box will appear. Type a file name in the **File Name** box. To change the location of the file, click the browse button next to the **File Name** box, browse and select the new location, and then click **OK**.



4. Click **Create**. Access creates the database and then opens an empty table (named **Table1**) in **Datasheet View**.

Datasheet View - A view that displays data from a table, form, query, view, or stored procedure in a row-and-column format. In Datasheet view, you can edit fields, add and delete data, and search for data.



From this point, the table structure can be created while you enter data. Any time that you add a new column to the table, a new field is defined. Access sets the data type.

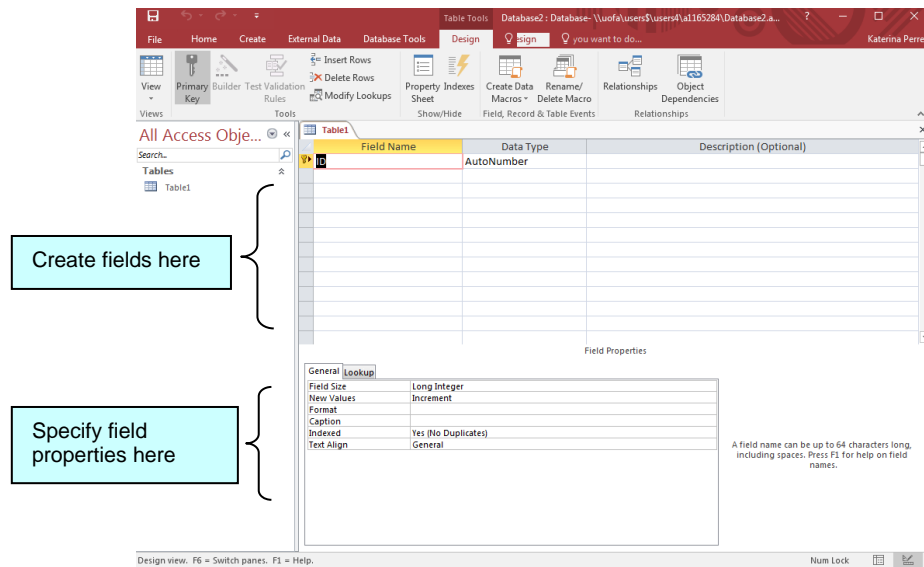
Data Type - The characteristic of a field that determines what type of data it can hold. E.g. text, number.

Creating a Table

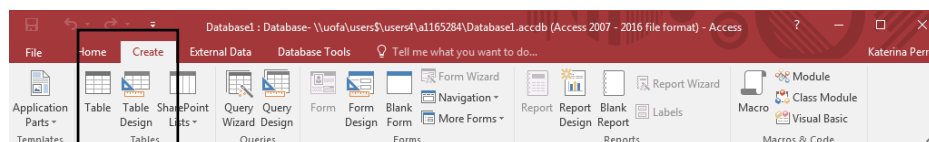
Before you create tables, carefully consider your requirements and determine all the tables that you need. Therefore, it is better to create your tables in **Design View** before entering your data.

When you first create a table, it is an empty container for data. You design the table to contain the specific type of data you want to store, such as names and addresses. After you save your table, it is ready for you to enter data.

1. Select **Design View**. You will be asked to give your table a name in the **Save As** dialog box. Enter a name and click **OK**. Microsoft Access opens the table window in **Design View**.



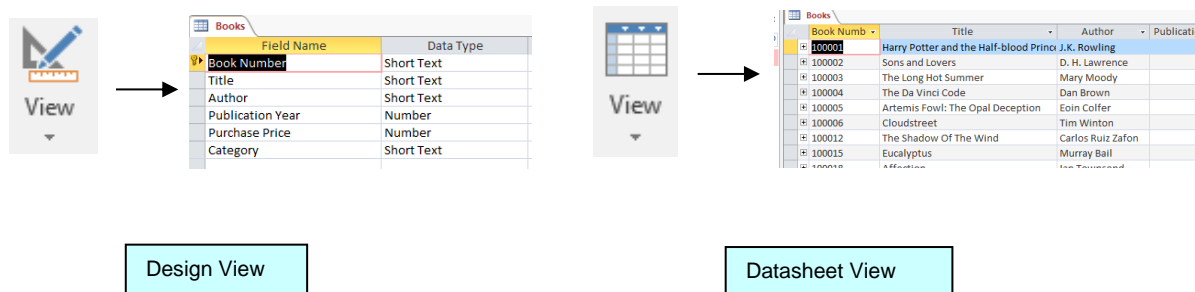
OR, if the database has already been created and you wish to create another table, click the **Create** tab and in the **Tables** group and select **Table**.



The Table Window

When you design a table, you specify the fields that you want it to contain in the upper portion of the table window. In the lower portion of the window, you can enhance your table by setting properties for each field.

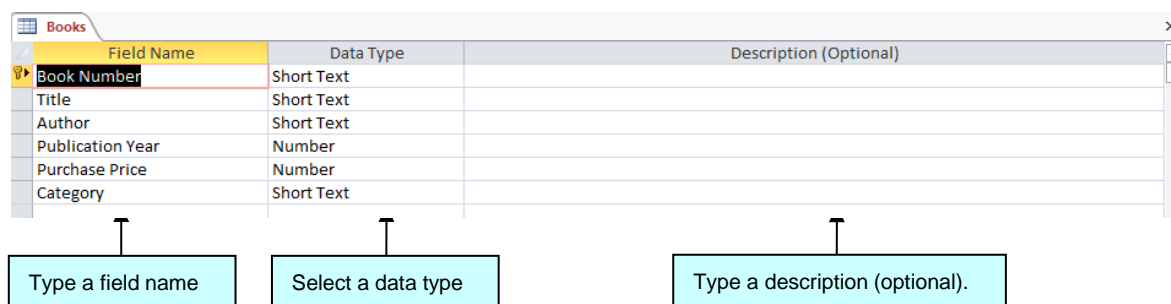
The buttons on the tool bar provide several shortcuts to commands you use when designing a table. For example, you can switch between viewing a table's structure and its data by clicking the **Design View** and **Datasheet View** buttons.



Adding Fields

When you create a table, you start by specifying fields in the first row of the upper portion of the table window. In each row, you enter the **Field Name**, **Data Type**, and field **Description** (optional).

Access automatically creates a primary key for you with every new table and assigns it a field name called "ID" with the **Autonumber** data type. You can change this.



Naming Fields

When you add a field, the first step is to type a name for it in the **Field Name** column. Field names can consist of up to 64 characters (letters or numbers), including spaces.

You should try to give field descriptive names so that you can easily identify them when viewing or editing records.

Creating a Primary key

1. In **Design View**, click on the field that requires a **Primary Key** (i.e. **Book number**) .
2. From the **Table Tools – Design** contextual tab, click on the icon **Primary Key** from the **Tools** group.

Field Name	Data Type
Book Number	Short Text
Title	Short Text
Author	Short Text
Publication Year	Number
Purchase Price	Number
Category	Short Text

A key will now appear to the left of the field chosen in the Field Name column

Choosing Data Types

After you name a field, you choose a data type for the data the field will store

When you add a field, Access automatically assigns it the **Text** data type. If you want the field to have a different data type, choose the data type you want from the drop-down list box in the **Data Type** column for the field.

Data Type
Text
Memo
Number
Date/Time
Currency
AutoNumber
Yes/No
OLE Object
Hyperlink
Attachment
Calculated
Lookup Wizard...

Describing Fields

In order to make your tables easier to understand and update, you can include a **Description** of each field in your table (optional). For example, if your table includes a currency field called **Price**, you might want to clarify what the field will store by entering "**Current retail price per unit**" in the **Description** column for the field.

When designing your tables, you should plan your tables carefully to avoid making changes later.

Field Properties in Access

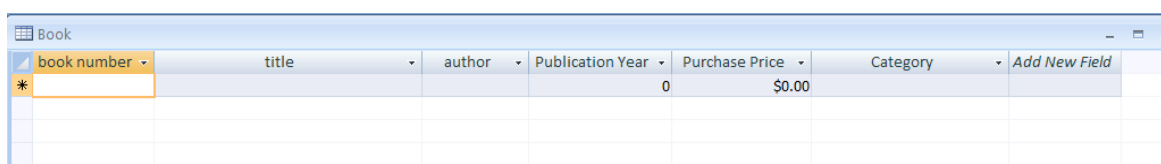
Fields are automatically assigned various properties by default depending upon their data type. These properties control the appearance and behaviour of the data. For example, by setting field properties in access we can prevent the incorrect data from being entered in a field, specify default values for a field or improve searching and sorting on a field.

The following table lists some of the field properties and their effects.

Field Size	Maximum number of characters or width	i.e. 50
Format	How the field is to be displayed	> (will show the text field as UPPERCASE) Medium date (change short date)
Input Mask	A template to allow ease of use in input	i.e. \(00\)\9999\ 9999 '9' number must be entered '0' number is optional
Caption	Another name for a field which will be displayed	i.e. First Name field can be shown as label Name in a table/query/report
Default Value	Automatic value. May be overridden	i.e. date()
Validation Rule	Logical expression which must be met by input data	i.e. >10
Validation Text	Text which is displayed when the validation rule is not met	i.e. Value entered must be greater than 10
Required	Is this field required to be filled	i.e. yes
Number Of Decimal Places	For numeric data, number of places to the right of the decimal point	i.e, 2

Adding Records

When you want to add records to a table you've finished designing, click the **Datasheet** button on the toolbar. Microsoft Access displays the table's datasheet, which shows fields for each record in the table in a tabular (row-column) format.



The screenshot shows the Datasheet view of a table named 'Book'. The table has seven columns: 'book number', 'title', 'author', 'Publication Year', 'Purchase Price', 'Category', and 'Add New Field'. The first row contains the values: an asterisk (*), an empty cell, an empty cell, the value '0', the value '\$0.00', an empty cell, and an empty cell. The 'book number' column is highlighted with a yellow background, and the first row is highlighted with a light blue background.

book number	title	author	Publication Year	Purchase Price	Category	Add New Field
*			0	\$0.00		

Exercise 2 - Data Types

- Fill in the table below for a member of a fitness club. For each field name, state the **Data Type**, **Field Description** and an appropriate **field size**.

Field Name	Data Type	Field Description	Field Size
Member ID			
First Name			
Surname			
Street			
Suburb			
Postcode			
Phone			
Date Of Birth			

- Open Microsoft Access and select **New** and **Blank database**.
- Enter **Fitness Centre** as the File Name and select the **U:** drive as the storage location. Click **Create**.
- Click **Design View** and save your table as **Members** to create a members table.
- Enter **Member ID** as the first field and as a primary key. Change the data type.
- Continue adding the remainder of the fields from the table above.
- Add the following members to your database.

aaa	Gail	Dunkley	105 Main St	Norwood	5067	89563256	12/04/72
bbb	George	Dunkley	105 Main St	Norwood	5067	89563256	05/11/70
ccc	Morgan	Coyne	12 Jetty St	Salisbury	5108	87512347	11/01/86
ddd	Rufus	Welch	58 Murphy Rd	Elizabeth	5112	85213571	28/12/90
eee	Sally	Wilson	34 Prince St	Glenelg	5045	85666658	04/04/65

- Close and save your table.

2. What are the three things you enter when creating a field?

3. Why do you need to select a field's data type?

4. Field descriptions are optional. What is the purpose of describing each field?

5. The following field names are of number data type for a Car Sales Database. Write an appropriate field size for each of the following field names. Use the example in the brackets to help determine the correct field size.

- | | |
|-----------------------------------|-------|
| a. Year (e.g. 2010) | _____ |
| b. Kilometres (e.g. 68 756) | _____ |
| c. Seats (e.g. 5) | _____ |
| d. Doors (e.g. 2) | _____ |
| e. Star Safety Rating (e.g. 3.5) | _____ |
| f. Towing Capacity ,Kg (e.g. 900) | _____ |

6. Using the previous example for a Car Sales Database, give an example of the following field properties for the field Seats.

- | | |
|--------------------|-------|
| a. Default Value | _____ |
| b. Validation Rule | _____ |
| c. Validation Text | _____ |
| d. Required | _____ |

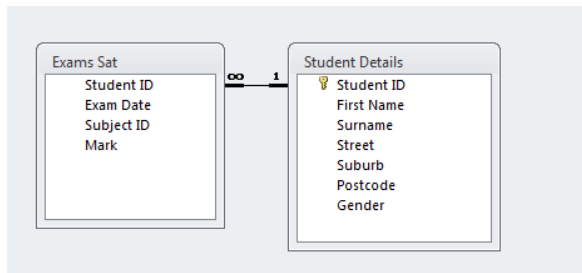
7. What is the purpose of a primary key?

Relational Databases

Data can be stored about different subjects in separate tables. To combine related data from separate tables you create relationships. A **relationship** is a logical connection between two tables that specifies fields that the tables have in common.

Relational databases are databases that store information in multiple tables instead of one. These tables are said to be interrelated.

Each table stores one specific category of information and shares at least one field with another table.



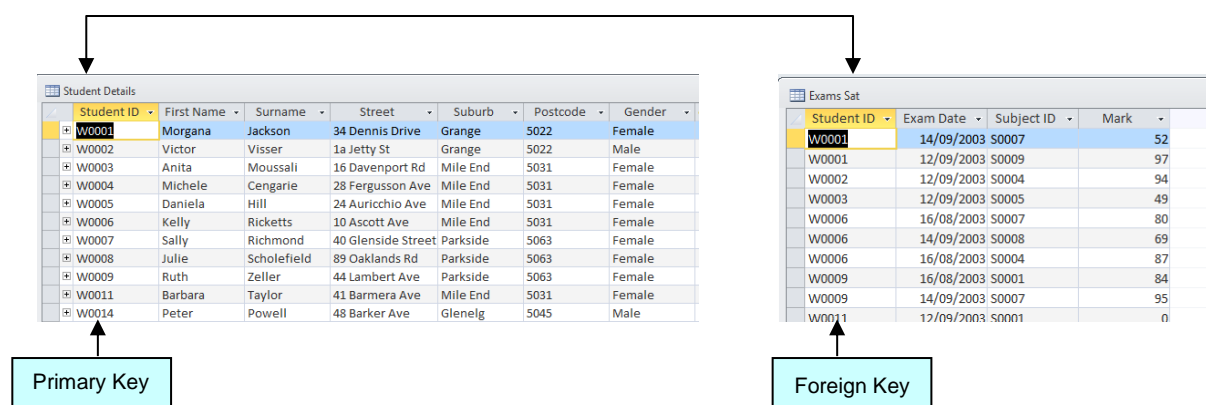
The line between these two tables shows there is a link (relationship) between these tables. This is done by linking the related fields

Relationships Between Tables

A common column, usually with the same name in both tables, forms a relationship. In most cases, the relationship matches the **primary key** from one table, which provides a unique identifier for each row, with an entry in the **foreign key** in the other table.

Relational databases support a number of different types of relationships between tables:

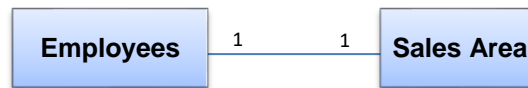
- **One – to – One** Relationship
- **One – to – Many** Relationship
- **Many – to – Many** Relationship



ONE-TO-ONE Relationships

A row in table A can only match one row in table B, and vice versa. A one-to-one relationship is created if both of the related columns are primary keys or have unique constraints.

E.g.



One Employee works in one Sales Area, and One Sales Area can only have one Employee

This type of relationship is not common because most information related in this way would be all in one table. You might use a one-to-one relationship to:

- Divide a table with many columns.
- Isolate part of a table for security reasons.

ONE-TO-MANY Relationships

A one-to-many relationship is the most common type of relationship. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A. Only one of the related columns is a primary key.

E.g.

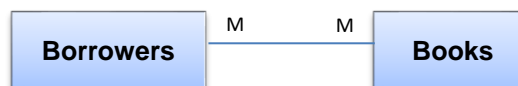


One Publisher produces many Book Titles, BUT one Book Title comes from only one Publisher.

MANY-TO-MANY Relationships

In a many-to-many relationship, a row in table A can have many matching rows in table B, and vice versa.

E.g. 1



One Borrower can borrow many books AND one Book can be borrowed by many Borrowers.

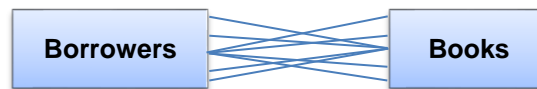
E.g. 2



One Student can take many Subjects AND one Subject can be taken by many Students.

The Problem with Many to Many Relationships

In relational database design, a many-to-many relationship is not allowed. Consider the example of keeping track of library books. If a borrower borrowed many books and a book was borrowed many times, your table design would look something like this:



If there were “many” books borrowed by the same Borrower Id, and the borrower asked to extend the loan of a book, how would you know which book they were referring to?

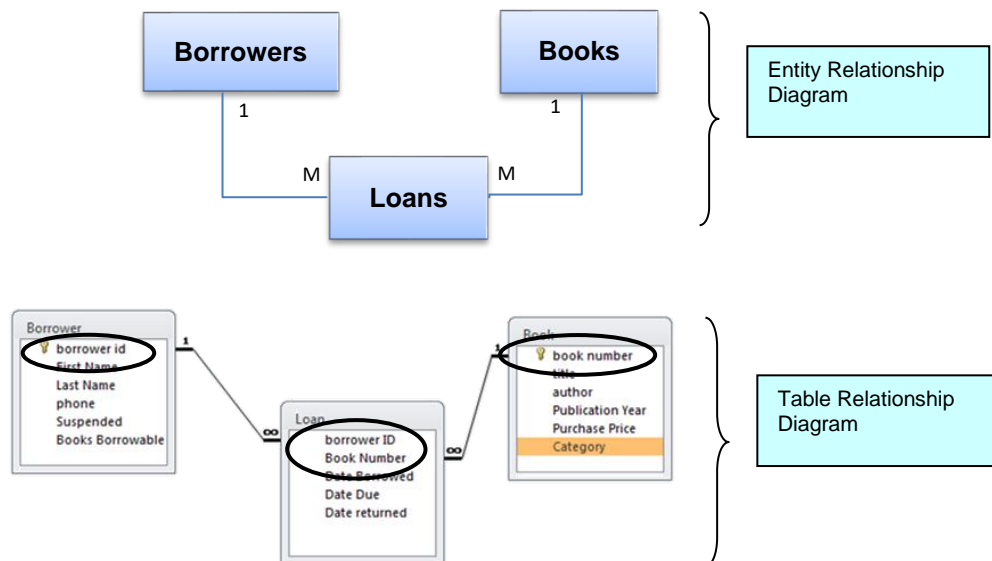
Transaction Tables

To resolve the problem of a many-to-many relationship you need to create a third table called a **transaction table**. Each record in the **transaction table** would have the foreign key fields of the two tables (**source tables**) it is joining together.

Let’s look at the previous example,



Solution:



A Borrower may Loan Many Books and a Book may have Many Loans

Adding the **Loan** transaction table solves the problem. The **Borrowers** table and the **Books** table is referred to as **Source tables**.

Database Management Systems (DBMS)

- Software that creates, manipulates and controls access to a database
- Software that manages relationships between several data tables
- Examples are Access, Oracle, and DB2

Advantages of DBMS

Reduced data redundancy	<p>Instead of the same data fields being repeated in different files, the information appears once. Therefore,</p> <ul style="list-style-type: none"> • No multiple record changes needed • More efficient storage • Simple to delete or modify details. • All records in other tables having a link to that entry will show the change. <p>Redundant data:</p> <ul style="list-style-type: none"> • Wastes disk space • Creates maintenance problems. <p><i>If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is much easier to do if that data is stored only in the Customers table and nowhere else in the database.</i></p>
Improved data integrity	<p>Reduced data redundancy increases the chances of data integrity:</p> <ul style="list-style-type: none"> • Data is accurate • Data is consistent • Data is up-to date – because each updating change is made in only one place. <p>In addition, database designers identify various integrity constraints during database design such as default values and validation rules, to improve data integrity.</p>
Complex queries can be created	<p>Far more complicated queries can be written that can extract data from many tables at once.</p>
Data security	<p>By splitting data into tables, certain tables can be made confidential. When a person logs on with their username and password, the system can then limit access only to those tables whose records they are authorised to view. For example, a receptionist would be able to view employee location and contact details but not their salary. A salesperson may see their team's sales performance but not competing teams.</p>
Centralised data management	<p>Allows multiple database users to access the data simultaneously</p>
Backup and recovery	<p>Recovery of data from hardware and software failures.</p>

Exercise 3 – Creating a Relationship

- Fill in the table below and then create a **Classes Table** for the **FITNESS CENTRE** database that you created in **Exercise 2** using Microsoft Access. For each field name, state the data type and an appropriate field size. Look at the sample data in part (c) to help you determine the data type.

Field Name	Data Type	Field Size
Class ID		
Class Name		
Length		
Instructor		
Location		

- Select **Class ID** as a primary key.
- Close and save the table as **Class**.
- Add the following “Classes” to your table.

111	Aerobics	50	Marie	Blue Room
112	Weights	30	Ben	Gym
113	Yoga	80	Rosa	Green Room

- Fill in the table below and then create a **Visits Table** for the fitness centre using Microsoft Access. For each field name, state the data type and an appropriate field size.

Note: For a relationship to be created, Member ID and Class ID are required to have the same data type and field size as set in the related tables. Unless the primary key field is an AutoNumber field and the foreign key field is a Number field, therefore the Field Size property of both fields needs to be the same.

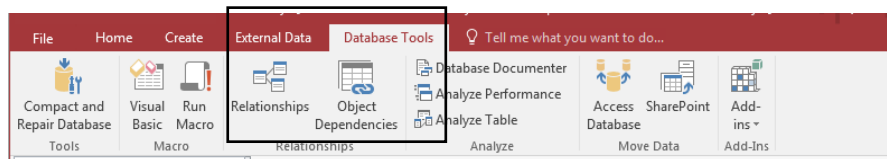
Field Name	Data Type	Field Size
Member ID		
Class ID		
Visit Date		

- No primary keys will be set in the **Visits** table so remove the primary key from the table.
- Close and save the table as **Visits**.

- c. Add the following “Visits” to your table.

aaa	111	3/10/12
bbb	111	3/10/12
ccc	111	3/10/12
ddd	111	3/10/12
aaa	112	4/10/12
eee	112	4/10/12
ccc	113	5/10/12

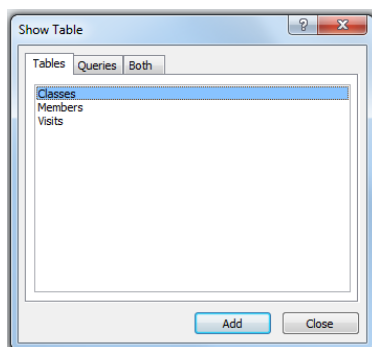
3. Select the Database Tools tab and choose Relationships from the Relationships group.



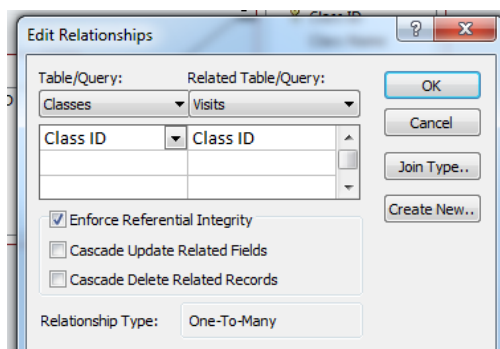
The **Show Table** dialog box automatically appears.

*(If it does not appear, select the **Design** contextual tab from the **Relationships Tools** tab and from the **Relationships** group, click **Show Table**.)*

4. Select one or more tables and then click Add. After you have finished adding all three tables to the Relationships document tab, click Close.



5. Drag the **Class ID** field (the primary key) from the **Classes** table to the **Class ID** field (the foreign key) in the **Visits** table. The **Edit Relationships** dialog box appears.
6. Select the **Enforce Referential Integrity** check box and click **Create**. Access draws a relationship between the two tables.



Enforce Referential Integrity

Enforce Referential Integrity ensures that a “Child” row can’t be added if the related “Parent” row doesn’t exist. That is, to make sure that we NEVER enter a record in one table (generally the transaction table) if the related record is non-existent in the source table. For example, don’t allow an Enrolment to be added for a non-existent Student.

This also asks Access to check on the records that may be left on their own when records are deleted from a table which is in a relationship with the records.

Once referential integrity rules have been selected, the following will NOT be allowed by Access:

- Deleting a record from the parent table
- Changing the primary key in a record which has related records
- Changing a key in the child table to a value that does not exist in the parent table.

7. Enter the following ‘Visit’ to your table:

Member ID	Class ID	Visit Date
fff	112	3/10/12

8. Explain what happens when you tried to add **Member Id “fff”** and their visit details in the database? Why did this happen?

Queries

A query allows you to answer a simple question, to perform calculations, to combine data from different tables, or even to add, change, or delete table data.

Queries allow you to:

- Sort records
- Choose only those fields you want to see
- Choose records that match your criteria
- Display data from several tables in a single datasheet
- Edit the data returned
- Create fields that generate calculated data

Criteria

In your design you should choose only the fields that contain data you want to display. This limits the number of columns in your result (the record set). Using criteria allows you to now limit the display of rows to only those records you want to see.

Examples of some criteria expressions:

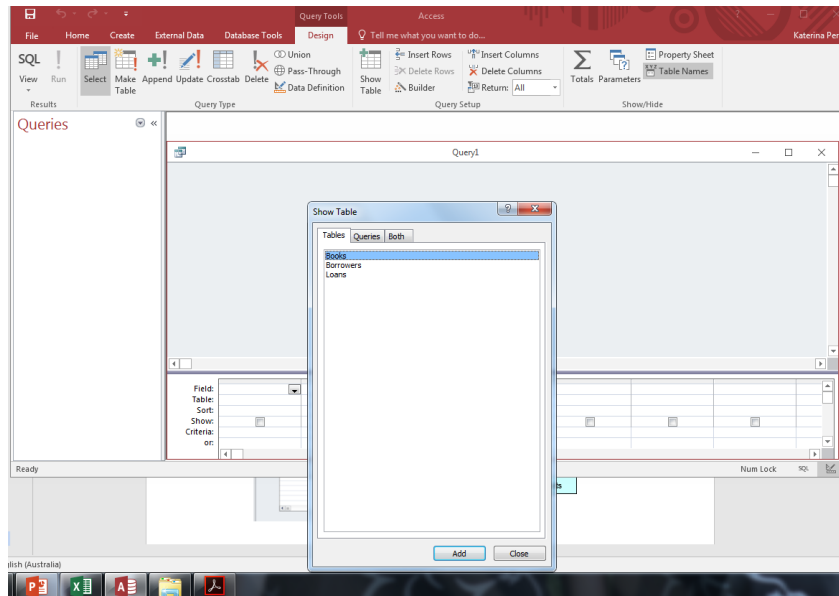
Field Name	Expression	Results
Title	"Daydream"	All records where the title exactly matches this.
Title	"Day*"	Returns all records where the title starts with Day.
Title	Like "[A-C]*"	Returns all records where the title starts with A, B, C.
Format	"CD" or "Vinyl"	Returns all records where the format was either CD or vinyl.
Postcode	Between "5000" and "5080"	Returns all records where the postcode is between 5000 and 5080 inclusive.
DateBought	<Date()-30	Returns all records where the date bought was more than 30 days ago.

Creating a Query

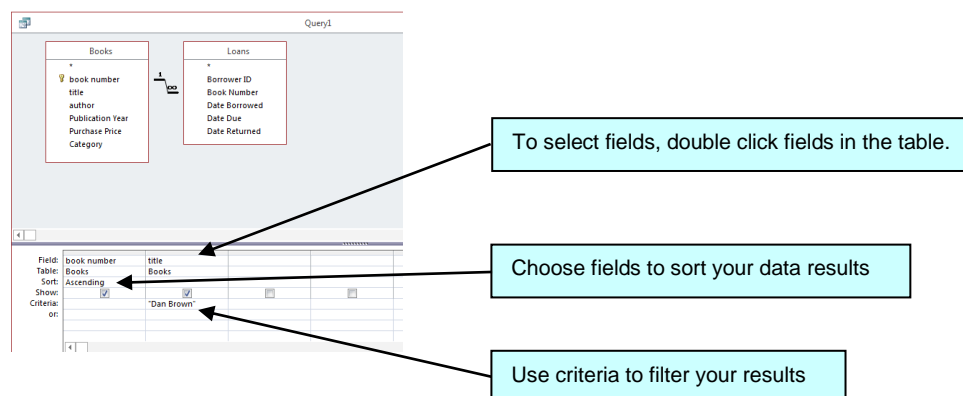
There are two ways to create queries in Access - choosing **Query Design** or **Query Wizard**. For this course we will be using **Query Design**.

1. Select the **Create** tab and click on **Query Design** from the **Queries** group.

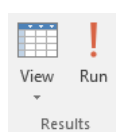
The **Show Table** dialog box automatically appears



2. Choose the table(s) that will be your record source (i.e. the tables you wish to query) and click **Add** then **Close**.
3. From each table, select the fields that you want to see in the query.



4. After you finish adding fields and any selection criteria, run your query to see if it gives you the correct results. Select **View** or **Run** from the Results group.

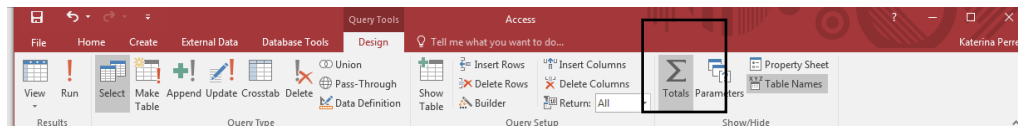


Creating a Statistical Query

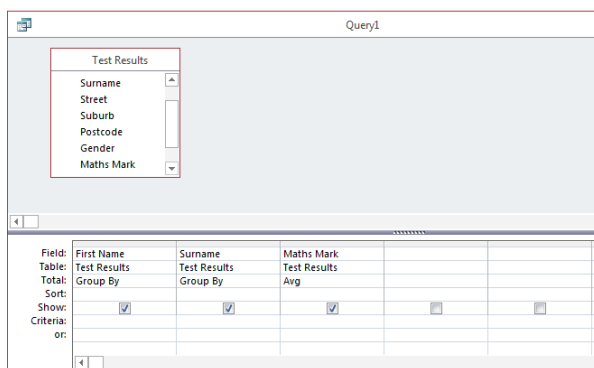
These queries require grouping on a field (or a number of fields), where fields that contain the same data are grouped together, so that a statistical calculation can be performed.

Performing an Average:

1. Select the **Totals** button:



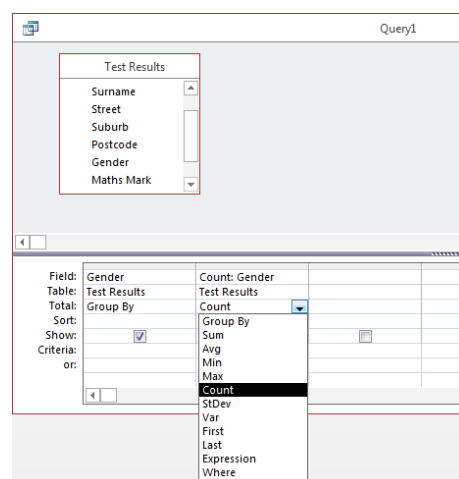
2. Next choose **Avg** from the dropdown menu



This query will display the average Maths Mark

Finding the Number of Occurrences:

1. Use **Totals** button, and select **Count**:



This query will display the number of each gender in the group.

Finding the maximum (**Max**), minimum (**Min**) and adding fields together (**Sum**) are done using the same methods.

Exercise 4 – Creating Queries

The following exercise refers to the **Library** database. Create the queries using Access and complete an **analysis** for each outcome (query).

1. Display the fields **Title**, **Author** and **Category** of books that have a category of fiction.

Outcome: A list of fiction books

Data Required: *SELECT Title, Author, Category FROM Books*

Processing: *WHERE Category = Fiction*

2. Display the fields **Title**, **Author**, **Purchase Price** and **Publication Year** of books that were published in 2003.

Outcome: _____

Data Required: _____

Processing: _____

3. Display the fields Title, Author, Purchase Price and Publication Year of books that were written by Tim Winton. The records should be sorted by ascending Title.

Outcome: _____

Data Required: _____

Processing: *WHERE* _____

ORDER BY Title

4. Display the first name, last name and phone number of all borrowers who are suspended. The records should be sorted by ascending last name.

Outcome: _____

Data Required: _____

Processing: *WHERE* _____

ORDER BY _____

5. Display the first name, last name and phone number of all borrowers who are allowed to borrow fewer than 4 books. The records should be sorted by descending phone number.

Outcome: _____

Data Required: _____

Processing: *WHERE* _____

ORDER BY _____

6. Display the title, author and date borrowed of all books that have been borrowed by Nicole Kidman. The records should be sorted by ascending date borrowed.

Outcome: _____

Data Required: _____

Processing: _____

7. Display the title, author and date borrowed of books that have been borrowed between the 5/6/2012 and 6/6/2012. The records should be sorted by ascending title.

Outcome: _____

Data Required: _____

Processing: _____

8. Display the first name, last name, title, author and date borrowed of books that either have been borrowed before May 2012 or that were written by J. K. Rowling. The records should be sorted by ascending title.

Outcome: _____

Data Required: _____

Processing: _____

9. Display the first name, last name, title, author and date borrowed of books written by J. K. Rowling, and that have been borrowed before May 2012. The records should be sorted by ascending title.

Outcome: _____

Data Required: _____

Processing: _____

10. Display the title, author and date borrowed of books where the author's surname is Moody.

Outcome: _____

Data Required: _____

Processing: _____

11. Display the title, author and price of the books that have a price between \$20 and \$30 or more than \$100.

Outcome: _____

Data Required: _____

Processing: _____

12. Display the title and author of books whose title starts with the letter "T". Sort by ascending author.

Outcome: _____

Data Required: _____

Processing: _____

13. Display the title of books borrowed in June 2011.

Outcome: _____

Data Required: _____

Processing: _____

14. Display the title of the non-fiction books that were due to be returned in June 2011.

Outcome: _____

Data Required: _____

Processing: _____

15. Display the titles of all books that do not start with the letter "T". Sort by ascending title.

Outcome: _____

Data Required: _____

Processing: _____

16. Display the book title, the borrowers' names, and the date due of all books that have not yet been returned.

Outcome: _____

Data Required: _____

Processing: _____

17. Display the first and last name of borrowers who borrowed a book more than 20 days ago.

Outcome: _____

Data Required: _____

Processing: _____

18. Display the first and last name of borrowers who borrowed a book in the last 10 days.

Outcome: _____

Data Required: _____

Processing: _____

19. Display the first and last name of borrowers who borrowed a book in May 2011 or May 2012.

Outcome: _____

Data Required: _____

Processing: _____

Exercise 5 – More Queries

Using the **Marks** database create the query in Access and complete an analysis for each of the following questions.

1. Display the total mark for each student (**Maths Mark + IT Mark**). Also display their first name, surname and suburb and sort by ascending surname. Save this query as **Total Mark**.

Outcome: _____

Data Required: *SELECT* _____

Processing: *WHERE [Maths Mark]+[IT Mark] AS [Total Mark]*
ORDER BY Surname

2. Display the average IT mark and the average maths mark for each suburb. Sort by ascending suburb.

Outcome: _____

Data Required: *SELECT* _____

Processing: *WHERE AVG([Maths mark] AND AVG([IT Mark])*
ORDER BY _____

3. Display the average IT mark and the average maths mark.

Outcome: _____

Data Required: _____

Processing: _____

4. Display the highest IT mark for each suburb. Sort by ascending suburb.

Outcome: _____

Data Required: _____

Processing: _____

5. Display the highest maths mark for females.

Outcome: _____

Data Required: _____

Processing: _____

6. Display the average maths mark for each exam date.

Outcome: _____

Data Required: _____

Processing: _____

7. Display the number of students who sat IT tests on 12/9/08.

Outcome: _____

Data Required: _____

Processing: _____

8. Display the sum of all the maths marks and IT marks for each gender.

Outcome: _____

Data Required: _____

Processing: _____

9. Display the average total mark for each suburb. Sort by ascending order of suburb.

Outcome: _____

Data Required: _____

Processing: _____

10. Display the first name, surname, maths mark and IT mark for a particular suburb. The user will enter the suburb required once the query has run. Sort by ascending order of surname. (**Use parameter box)

Outcome: _____

Data Required: _____

Processing: _____

Exercise 6 – Written Exercises

1. What is a database?

2. List three advantages of using a database.

3. What is meant by the term “redundant data”?

4. What is a “key field”?

5. Describe what is meant by a one-to-many relationship? Give an example.

6. Describe what is meant by a many-to-many relationship? Give an example.

7. What is the purpose of a transaction table? Give an example.

8. A default value is _____

9. A validation rule is _____

Exercise 7 – Field Properties

1. a. Examine the data below.

First	Last	Title	Address	City	State	Post Code	Phone	Number of Patients
Mark	Garver	M.D.	17053 South 71 Highway	Belton	SA	5401	(08) 555-4000	156
Richard	Hambley	M.D.	17053 South 71 Highway	Belton	SA	5401	(08) 555-4000	98
Doug	Braden	M.D.	204 East North Avenue	Belton	SA	5401	(08) 555-7900	127
Kirk	Ridwell	M.D.	206 East North Avenue	Belton	SA	5401	(08) 555-7744	200
Gary	Gomez	D.O.	8015 East 171st Street	Belton	SA	5401	(08) 555-0961	178
William	Baker	D.O.	12121 Blue Ridge Blvd	Grandview	WA	6403	(08) 555-0884	194
Samuel	Harley	M.D.	6420 Prospect	Kansas City	NSW	2412	(02) 555-4100	123
Kirk	Barnhart	M.D.	900 Main St.	Grandview	WA	6403	(08) 555-8900	453
Vicki	Kowalewski	M.D.	13010 White Avenue	Grandview	WA	6400	(08) 555-1080	368
Jeffrey	Simmons	M.D.	6215 Main Street	Grandview	WA	6403	(08) 555-3888	176
John	Collins	M.D.	6901 W. 121st St.	Overland Park	NSW	2629	(02) 555-7886	278

- b. Complete the following table for each of the above fields.

Field Name	Data Type	Field Size	Default Value	Validation rule
First				
Last				
Title				
Address				
City				
State				
Postcode				
Phone				
Number of Patients				

2. Examine the data on the following page for the Healthy Fitness Centre and determine the most appropriate field names, data type, sizes, validation rule and default values.

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Field name	
Data Type	
Field Size	
Default value	
Validation Rule	

Healthy Fitness Centre

Membership Number: 0109
Name: Judith Warrick
Contact phone: 8296 1126
Date Joined: 15/5/03
Membership Type: Junior
Adult
Senior
Dues paid: Yes / No
Number of activities: 2

Healthy Fitness Centre

Membership Number: 0259
Name: Sarah Henderson
Contact phone: 0423 111 223
Date Joined: 15/6/03
Membership Type: Junior
Adult
Senior
Dues paid: Yes / No
Number of activities: 3

Healthy Fitness Centre

Membership Number: 1267
Name: William Tell
Contact phone: 0411 234 567
Date Joined: 30/6/01
Membership Type: Junior
Adult
Senior
Dues paid: Yes / No
Number of activities: 2

Database Normalisation

Normalisation is the process of organising data in a database to minimise redundancy. This includes creating tables and establishing relationships between those tables. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then those changes are made through the rest of the database via the defined relationships

Normalisation principles are:

- Eliminating redundant data (i.e. storing the same data in more than one table)
- A primary key consists of one or more fields that uniquely identify a record in a table
- Non-key fields are dependent on the key (Eliminate fields that do not depend on the key, i.e. a book borrowed by a borrower should not appear in the Borrower table)
- Non-key fields are not dependent on another field (except the key)

Normalising an Example Table

These steps demonstrate the process of normalising a fictitious student table.

Unnormalised table:

Student ID	Bachelor	SATAC Code	Subject 1	Subject 2	Subject 3
1022	Computer Science	314111	MASF	CIS	EAP
4123	Computers Science	314111	MASF	CTHF	CIS
1011	Commerce	314101	CTHF	CIS	MASF

First Normal Form (1NF):

No Repeating Groups. Eliminate duplicative columns from the same table.

Each field should have different data. Since one student takes several subjects (one-to-many relationship), these subjects should be listed in a separate table. Fields Subject1, Subject2, and Subject3 in the above records will cause many problems.

Problems:

- "What if a student only does one subject"? - Then the other fields are wasted space.
- "what if a student does a 4th subject? - Then the entire table structure will require modification.

Another way to look at this problem is with a one-to-many relationship. Do not put the one side and the many side in the same table. Instead, create another table in first normal form by eliminating the repeating group (Subject#), as shown below:

Student ID	Bachelor	SATAC Code	Subject ID
1022	Computer Science	314111	MASF
1022	Computer Science	314111	CIS
1022	Computer Science	314111	EAP
4123	Computers Science	314111	MASF
4123	Computers Science	314111	CTHF
4123	Computers Science	314111	CIS
1011	Commerce	314101	CTHF

Second Normal Form (2NF):

Eliminate Redundant Data

Note the multiple **Subject ID** values for each **StudentID** value in the previous table. This causes the redundant data because the Student ID, Bachelor and SATAC Code are repeated.

Solution: Remove the data that appears in multiple rows of a table and place them in a separate table. Then create relationships between those tables.

The following two tables demonstrate second normal form:

STUDENT Table:

Student ID	Bachelor	SATAC Code
1022	Computer Science	314111
4123	Computers Science	314111
1011	Commerce	314101

STUDENT CLASS Table:

Student ID	Subject ID
1022	MASF
1022	CIS
1022	EAP
4123	MASF
4123	CTHF
4123	CIS
1011	CTHF

Third Normal Form: Eliminate Data Not Dependent On Key

In the last example, **SATAC Code** is dependent on the **Bachelor** attribute. The solution is to move that attribute from the **Students** table to the **Degrees** table, as shown below:

STUDENTS Table:

Student ID	Bachelor
1022	Computer Science
4123	Computers Science
1011	Commerce

DEGREE Table:

Bachelor	SATAC Code
Computer Science	314111
Commerce	314101

Exercise 8 – Database Normalisation

1. Consider the **Fitness Centre** database you created using Access in **Exercise 2**. The table below shows how the fitness centre first kept a list of its members' details and their visits to the health centre in the following table. Examine the data and answer the following questions.

First Name	Surname	Street	Suburb	Phone	Date Of Birth	Age	Membership Category	Membership Fee	Visit Date	Activity	Class Length (min)	Instructor	Location
Gail	Dunkley	105 Main St	Downtown	89563256	12/1/1990	14	6-day	\$135	6/6/2012	Aerobics	50	Marie	Blue Room
George	Dunkley	105 Main St	Downtown	89563256	22/1/1969	35	Full	\$200	6/6/2012	Weights	30	Ben	Gym
Morgan	Coyne	12 Jetty St	Salisbury	87512347	4/1/1964	40	Full	\$200	7/6/2012	Weights	30	Julie	Gym
Rufus	Welch	58 Murphy Rd	Elizabeth	85213571	15/6/1983	21	Full	\$200	7/6/2012	Weights	30	Ben	Gym
Morgan	Coyne	12 Jetty St	Salisbury	87512347	4/1/1964	40	Full	\$200	8/6/2012	Yoga	80	Rosa	Green Room
Meagan	Davis	6 Mountain Rd	Salisbury	87445223	15/8/1989	15	6-day	\$135	8/6/2012	Yoga	80	Rosa	Green Room
Gail	Dunkley	105 Main St	Downtown	89563256	12/1/1990	14	6-day	\$135	8/6/2012	Aerobics	50	Marie	Blue Room
Sally	Wilson	34 Prince St	Glenelg	85666658	31/1/1989	19	6-day	\$135	8/6/2012	Yoga	80	Marie	Green Room

- a. Identify two different examples of **data** from this table.

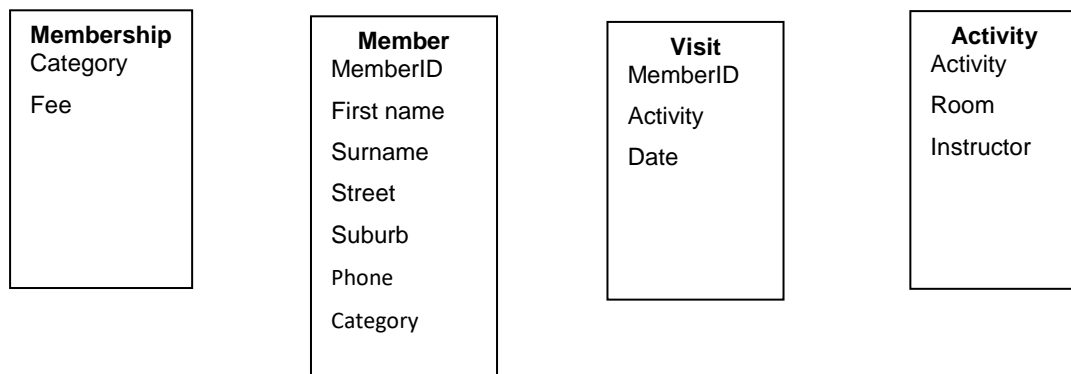
- b. Highlight and label a record.

- c. Highlight and label a field.

- d. The Age field was stored as a 'number' data type. Suggest why this would cause a problem in the database.

- e. Identify two different examples of redundant data.

- f. The following is one possible way that the tables could be implemented in a relational database. Underline the keys. Draw the relationships between these tables, and label the relationships.



- g. However there is an error in this layout. Look closely at the data again. Note that Yoga is taught by two different teachers. Suggest a solution to the error.

2. A doctor's surgery keeps a list of its patient details in the following table. Examine the data and answer the following questions.

First Name	Surname	Phone	Date of birth	Age	Doctor Name	Doctor Phone	Surgery Room	Visit Date	Visit details
Gail	Dunkley	84555556	12/1/1990	14	Martin	87775635	R1	15/5/2012	Virus
George	Dunkely	84555556	22/1/1969	35	Martin	87775635	R1	15/5/2012	Back ache
Morgan	Coyne	86548789	4/1/1964	40	Jones	85692211	R3	16/6/2012	Flu
Rufus	Welch	85699987	15/6/1983	21	McGregor	87563322	R2	16/6/2012	Flu
Kristian	Flach	89862323	4/5/1939	65	Martin	87775635	R1	17/6/2012	Sore Knee
Gail	Dunkley	84555556	12/1/1990	14	McGregor	87563322	R2	18/6/2012	Broken arm
Rufus	Welch	85699987	15/6/1983	21	Jones	85692211	R3	19/6/2012	Check up

- a. Identify two different examples of data from this table. _____
- b. By using the data in the table, identify information that shows processing. _____
- c. Identify two different examples of redundant data. _____

- d. Draw tables that demonstrate how this information could be broken up so that there is no data that is repeated unnecessarily (i.e. to eliminate redundant data).

- e. What would be suitable **keys** for the tables?

- f. On the diagram, draw the relationships between the tables and show the type of relationship.

- g. Which table is the transaction table for this database? _____

- h. Describe the many-to-many relationship in these tables.

3. A car yard sells and services cars. It keeps a list of the cars that have been serviced so far this year and the mechanic who serviced the car in the following table. Examine the data and answer the following questions.

Owner First Name	Owner Surname	Owner Phone	Car Registration	Date Bought	Years owned	Mechanic	Workshop	Date of service
Gail	Dunkley	82234455	YRE 123	12/1/2007	5	Bloggs	Blue	15/1/2012
Gail	Dunkely	82234455	YRE 123	12/1/2001	5	Bloggs	Blue	31/3/2012
Morgan	Coyne	81125891	WDE 457	4/1/2011	1	Calvin	Green	8/4/2012
Rufus	Welch	86655233	RFD 332	15/6/2011	1	Bloggs	Blue	19/6/2012
Kristian	Flach	85561124	YYE 979	4/5/2011	1	Tilburgs	Green	7/6/2012
Morgan	Coyne	81125891	FRE 523	15/8/2007	5	Tilburgs	Green	23/5/2012
Rufus	Welch	86655233	RFD 332	15/6/2011	1	Calvin	Green	1/5/2012

- a. By using the data in the table, identify information that shows processing.

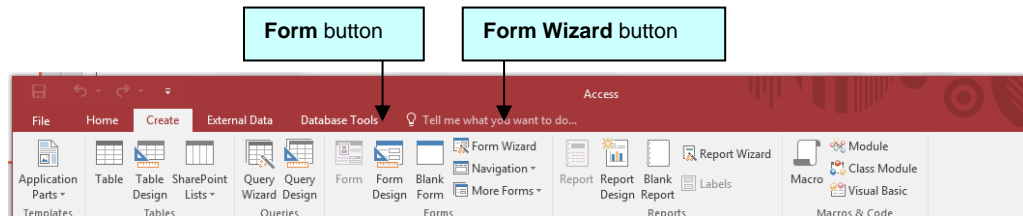
- b. Identify two different examples of redundant data.

- c. Show, by drawing the tables, how this information could be broken up so that there is no unnecessary repeated data.

- d. What additional data could usefully be stored for this business?

Forms

Forms are used to display information onto the computer monitor and are commonly used for data entry of data into tables. They can display a number of related tables or a single table. Forms can contain buttons (for example, to display other forms or reports) and text boxes to input or select data.



Creating a Simple Form

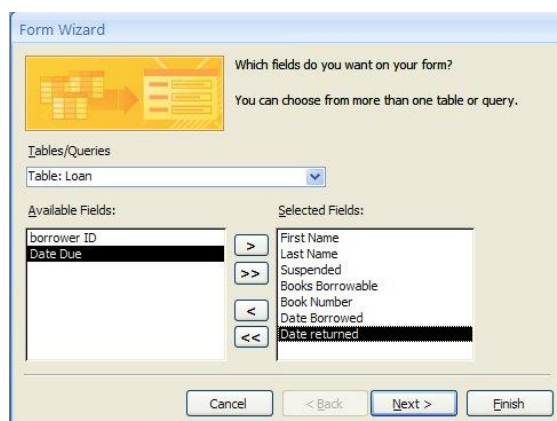
1. Select the **Create** tab.
2. Click on the table you want to create a form for.
3. Click on the **Form** button from the **Forms** group. Adjust the form elements to suit your needs then save the form.

Creating a More Complex Form

1. Select the **More Forms** button to start the **Form Wizard**.
2. To create a form that uses a transaction table but gets data from other tables so that the user does not need to look up different details, use the **Form Wizard** to create a form with subforms.

Exercise 9 - Forms

1. Open the **Library** database and click on the **Create** tab.
2. Click **Form Wizard**.
3. Select the **Borrowers** table and choose **First Name**, **Last Name**, **Suspended** and **Books Borrowable**.
4. Select the **Loan** table and choose **Book Number**, **Date Borrowed** and **Date Returned**. Click **Next**.



5. Next you will be asked: **How do you want to view your data?** Make sure the **Form with subform(s)** button is checked and that the layout template looks similar to the picture. Click **Next**.

Form Wizard

How do you want to view your data?

by Borrower
by Loan

First Name, Last Name, Suspended, Books Borrowable

Book Number, Date Borrowed, Date returned

☒ Form with subform(s) ☐ Linked forms

Cancel < Back Next > Finish

6. Next you will be asked: **What layout would you like for your subform?** Make sure the **Datasheet** button is checked. Click **Next**.

Form Wizard

What layout would you like for your subform?

☐ Tabular ☒ Datasheet

Cancel < Back Next > Finish

7. You will then be asked: **What titles do you want for your forms?** Type the name you want for the form in the **Form** box i.e. **Borrowers Loans** and choose **Modify the form's design**. Click **Finish**.

Form Wizard

What titles do you want for your forms?

Form: Borrower Loans

Subform: Loan Subform

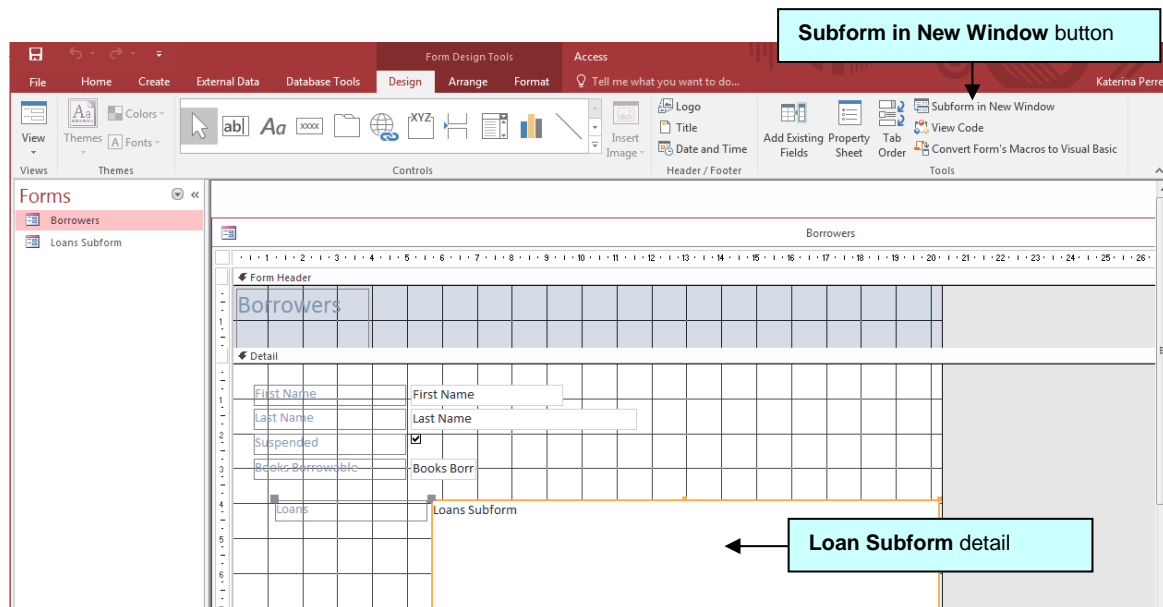
That's all the information the wizard needs to create your form.

Do you want to open the form or modify the form's design?

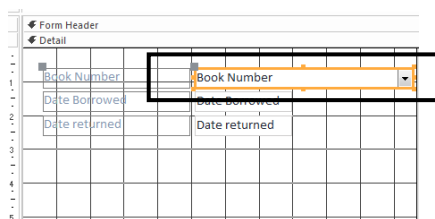
☐ Open the form to view or enter information. ☒ Modify the form's design.

Cancel < Back Next > Finish

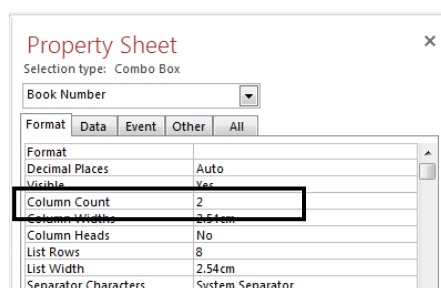
8. Make sure that you are in the **Design View** and click to select the **Loans Subform** detail.



9. Click **Subform in New Window** from the **Tools** group.
10. Click the **Book Number** text box and make sure it is highlighted.



11. Right click and choose **Properties** from the menu.
12. In the **Format** tab change the **Column Count** to 2.

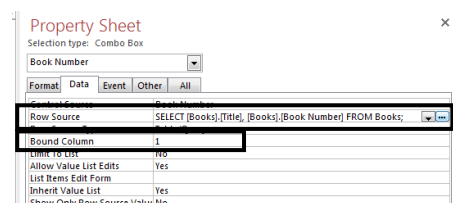


13. Select the **Data** tab and on the line that says **Row Source**, change the text to read

SELECT [Books].[Title], [Books].[Book Number] FROM Books;

Make sure you have the semi colon at the end of the sentence.

14. Change the Bound Column from 1 to 2.



15. Save the changes and select the **Form View** from the **Home** tab to view the **Borrowers Form**.
You should now see the title of the books that each person borrowed.

Borrowers

First Name: Cathy
Last Name: Davies
Suspended: ☐
Books Borrowable: 4

Loans

Book Number	Date Borrowed	Date returned
Harry Potter and the Half-blood Prince	6/06/2011	25/06/2011
Artemis Fowl: The Opal Deception	6/06/2011	26/06/2011
The Long Hot Summer	6/06/2011	22/06/2011
The Da Vinci Code	1/05/2011	6/06/2011
Cloudstreet	10/06/2011	28/06/2011
The Turning	6/06/2012	
*	20/01/2013	

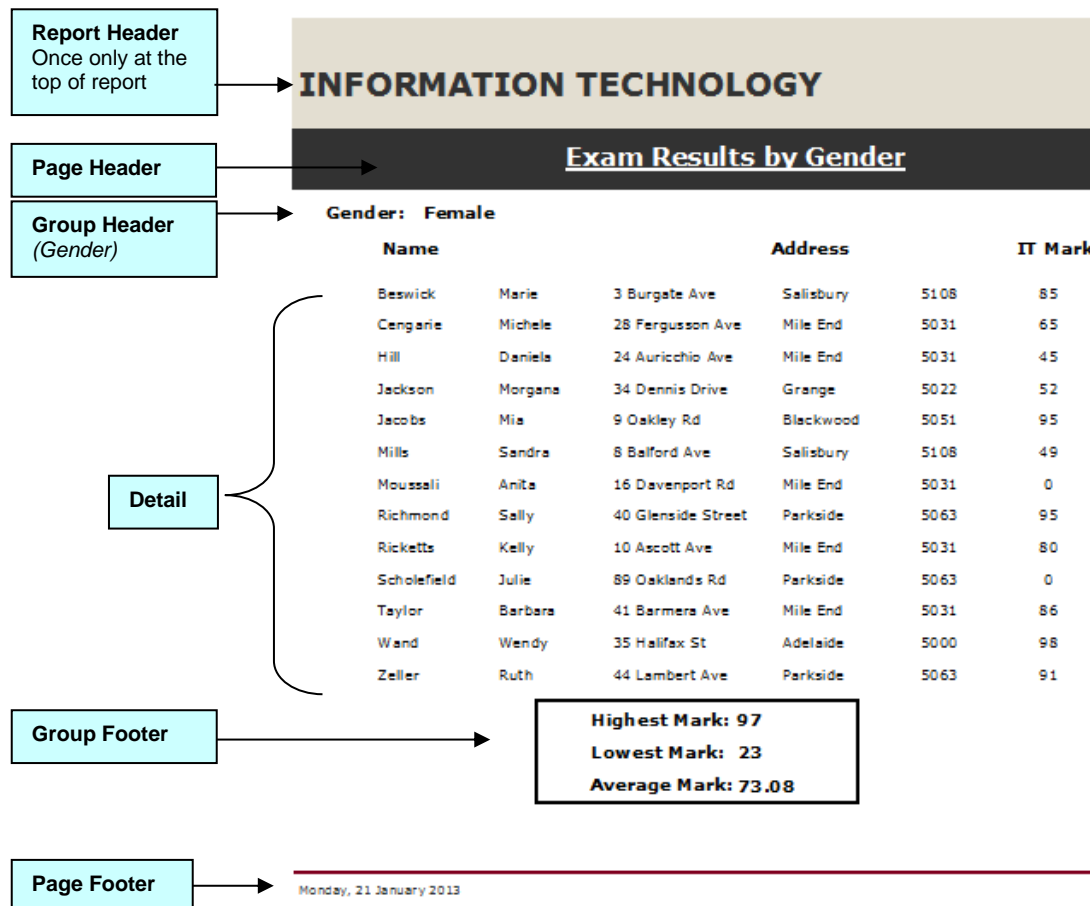
Record: 1 of 9

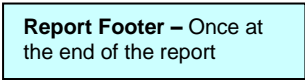
Reports

Information is made user-friendly by designing **reports**.

Reports are used to display the output from a query or some data from related tables in a printable format. Reports are also the best way to format and print your data, and they're a good way to summarise data. Fields can be grouped and calculations can be added to a report. The data can be formatted in many ways to suit your requirements.

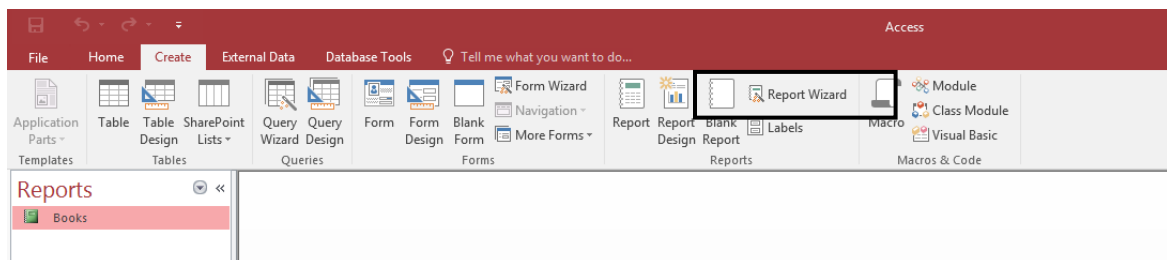
Parts of a Report





The **Report Wizard** is a feature in access that leads you through a series of questions/steps and then produces a report based on your answers. With a report wizard you can select what fields appear on your report, how the data is grouped and sorted, and you can use fields from more than one table or query.

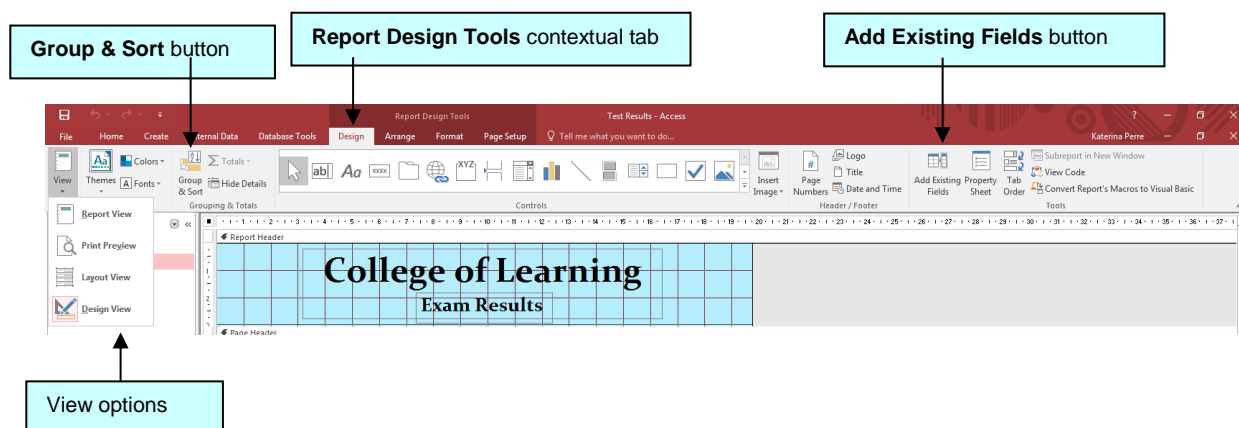
3. Follow the directions on the **Report Wizard** pages. On the last page, click **Finish** to view your report.



Exercise 10 - Reports



1. Open the **Marks.accdb** database.
2. Create a report, using the **Report Wizard**, that contains the following details (a sample is on the next page):
 - a. Select ALL the fields from the **Test Results** table. Click **Next**.
 - b. Group by **Exam Date** and then by **Gender**.
*Note: Click **Grouping Options** to change the **Grouping Interval** from **Monthly** to **Normal** for the **Exam Date**.*
 - c. Click **Next**.
 - d. Sort the **Surname** in ascending order. Click **Next**.
 - e. Choose a **Stepped** layout. Click **Next**.
3. Change the title to **Exam Results**. Click **Finish**.
4. Examine your report in **Print Preview**. Click **Close Print Preview** and format the report so that all the fields are visible and aligned neatly. You can format your report by using **Layout View** or **Design View** from the **Design** tab which can be found in the **Report Design Tools** contextual tab.



You can use **Add Existing Fields** to add any fields that are missing from your report. A list of fields from your table will be displayed. Drag the field onto your report.

5. Change the headings to **Name** and **Address**.
6. Include a major heading of **College of Learning**. This is to be positioned at the top of the report in the report header.
7. Add the average of each exam mark at the end of the report. This will be placed in the report footer. Into a text box, place the formula

$$=Round(Avg([Maths Mark]),2)$$
8. Under the **Maths Marks** add a similar formula for the average IT mark.
9. Include a heading for these averages of: **Overall Averages**
10. Add the average of each exam mark under the gender grouping. This will be placed in the **Gender** footer.
11. If this is not visible in your report, you need to show it by selecting **Layout View** or **Design View** from the **Design** tab which can be found in the **Report design Tools** contextual tab.
12. Click **Group & Sort** from the **Grouping & Totals** group. Select the **Gender** field and change the **Group Footer** to 'yes'. Add the average formulas as above to this section.

13. Include a heading for these averages of: **Average for Gender**
14. Your report should look similar to the one on the next page.

College of Learning Exam Results							
Exam Date	Gender	Name	Address		Maths Mark	IT Mark	
14/08/2007	Female	Jackson	Morgana	34 Dennis Drive	Grange	5022	65 52
Average for gender:						65	52
12/09/2012	Female	Cengarie	Michele	28 Fergusson Ave	Mile End	5031	49 65
		Moussali	Anika	16 Darenport Rd	Mile End	5031	89 0
		Taylor	Barbara	41 Barmra Ave	Mile End	5031	82 86
		Zeller	Ruth	44 Lambert Ave	Parkside	5063	75 91
Average for gender:						73.75	61
	Male	Andrill	Jonathon	2 Leeshman Court	Norwood	5067	0 94

Exam Date	Gender	Name	Address		Maths Mark	IT Mark	
16/11/2012	Female	Bestwick	Marie	3 BURGATE AVE	Salisbury	5108	69 85
		Jacobs	Mia	9 Oakley Rd	Blackwood	5051	87 95
		Mills	Sandra	8 Balford Ave	Salisbury	5108	78 49
		Richmond	Sally	40 Glenside Street	Parkside	5063	94 95
		Ricketts	Kelly	10 Ascott Ave	Mile End	5031	86 80
Average for gender:						82.8	81
	Male	Powell	Peter	48 Barker Ave	Glenelg	5045	76 84
		Visser	Victor	12 Jetty St	Grange	5022	85 94
Average for gender:						80.5	89
Overall Averages							
Average Maths Mark					69.86		
Average IT Mark					72.04		

15. Start each exam date details on a new page.

To do this: Select the **Exam Date** footer and click **Property Sheet** from the **Tools** group. Select **Force New Page** and choose **Before Section** from the drop down box.

16. Print your report for your **PORTFOLIO**.

Review Exercise 1: Company

1. Open the database: **Company**
2. Examine the data and change the field sizes to more appropriate sizes on:
 - a. **Employees** table: **first name, surname**
 - b. **Jobs** table: **project**
3. In the **Jobs** table, give the field **Number Hours** a default value of **10** and an appropriate validation rule (as shown by the validation text).
4. In the **Positions** table, give the field **Pay Rate** an appropriate validation rule (as shown by the validation text).
5. Create a query called **Mechanics** which displays the names of the mechanics. Sort by surname.
6. Create a query called **ProjectA** that displays the employees name and their pay rate for employees who have worked on project A for more than 20 hours. Sort this by surname.
7. Create a query called **Elizabeth** that displays the employee names of employees that live in a suburb whose name contains the word Elizabeth.
8. Create a query called **Postcodes** that displays the suburb and the postcode of those postcodes between 5022 and 5045 inclusive. Sort the suburb into descending order.
9. Create a form to enter the job details for an employee.

Project	Number Hours	Week ending
A	15	3/06/2005
B	19	3/06/2005
A	25	10/06/2005
B	20	10/06/2005

10. Create a report like the one shown on the next page. At the end of each employee, show the total number of hours worked. At the end of the report, show the average number of hours worked for all employees.

Figure 1 shows the detail of the first two employees on the first page. **Figure 2** shows the final report layout.

Hours Worked				
Name	Position	Project	Week Ending	Number Hours
Fred Bloggs	Supervisor	A	3/06/2005	15
			10/06/2005	25
		B	3/06/2005	19
			10/06/2005	20
Total hours for employee:				79
Mike Hutchinson	Electrician	A	3/06/2005	21
			10/06/2005	30
		B	3/06/2005	25
			10/06/2005	17
Total hours for employee:				93
Betty Buster	Mechanic	A	3/06/2005	10
			10/06/2005	10
		B	10/06/2005	21
Total hours for employee:				41
David Donaldson	Electrician	A	3/06/2005	19
			10/06/2005	10
		B	3/06/2005	29

First page

Name	Position	Project	Week Ending	Number Hours
		B	3/06/2005	23
			10/06/2005	37
Total hours for employee:				101
Frank Miles	Electrician	A	3/06/2005	32
			10/06/2005	35
		B	3/06/2005	25
			10/06/2005	2
Total hours for employee:				94
Susan Jones	Mechanic	A	3/06/2005	37
		B	3/06/2005	21
			10/06/2005	22
Total hours for employee:				80
Average hours worked:			23.1428571428571	

Final Page

Review Exercise 2: Media Loft

1. Open the database: **MediaLoft**
 - a. Examine the data to complete the following questions:
 - b. Change the design of the table to include the correct field size for each field in the table
 - c. Change the field properties to match that outlined in the table below:

Field name	Default Value	Validation rule
Category	New Age	
Format	CD	CD, Vinyl, Cassette are only valid values
Tracks	6	Greater than 0
Wholesale	7	Greater than 5
Retail	15	Greater than 10

2. Create a query that displays all the records of category jazz that are on cassette. Display the title, artist and category. Sort the record in order of title.
3. Create a query that displays the title, artist, format, wholesale price and tracks, sorted by artist, of all records where the wholesale price is greater than \$9 and which also have less than 8 tracks.
4. Create a query that calculates the tax owing on each record. The tax is calculated as 20% of the retail price. Display the title, artist, wholesale price and tax, sorted by title and then artist.
5. Create a query that displays the average retail price of each category. (HINT the only fields that should be displayed are each category and the average retail price.)
6. Create a form for the **Music Inventory** table. Place the name of the table in the header of the form. Also include your name in the header of the form. In the form footer include a count of the total number of records for the table.
7. Create a report that displays the title, artist, label, format and quantity, grouped by category.
 - a. Display the total titles for each category and at the end of the report.
 - b. Display the records in ascending alphabetical order of title.
 - c. The heading should be: **Quantity Of Titles For Each Category.**
 - d. Include a clipart of your choice in the report.
 - e. Ensure that the report is formatted so that it is well presented – use different fonts and colour for the headings.
 - f. The bottom of each page should contain your name to the left and the date to the right.

Quantity of Titles For Each Category				
Category	Title	Artist	Label	Format
Children	Raffi in Concert	Raffi	ABC Records	Cassette
				Total: 1
Classical	Fantasia	Stokowski, Leopold	Buena Vista Records	CD
	Favorite Overtures	Bernstein, Leonard	CBS Records	Vinyl
	Handel's Messiah	Bernstein, Leonard	CBS Records	Cassette
				Total: 3
Country	Christmas to Christmas	Greenwood, Lee	MCA Records	CD
	Garth Brooks Live	Brooks, Garth	Liberty Records	CD
	The Chase	Brooks, Garth	A&M Records	CD
	The Song Remembers When	Yearwood, Trisha	MCA Records	CD
				Total: 4
Folk	A Christmas Album	Grant, Amy	Reunion Records	CD
	Can We Go Home Now	Roches, The	Rykco	CD
	Heart in Motion	Grant, Amy	A&M Records	Cassette
	The Roches	Roches, The	Warner Bros. Records	Cassette
				Total: 4
Gospel	Come Walk With Me	Adams, Oleta	CBS Records	CD
	Greatest Hits	Winans, BeBe & CeCe	Benson	Vinyl
	I'll Lead You Home	Smith, Michael	Reunion	CD
	Message	4 Him	Benson	CD

First page

				Total: 19
Rap	God's Property	Nu Nation	B-Rite Music	Cassette
	World Café	Tree Frogs	New Stuff	CD
				Total: 2
Rock	Abbey Road	Beatles, The	Capitol Records	Vinyl
	Blue	Mitchell, Joni	Liberty Records	CD
	Cosmic Thing	B-52s	Warner Records	CD
	Cracked Rear View	Hootie and the Blowfis	Arista	CD
	Daydream	Carey, Mariah	Columbia	CD
	Decade	Young, Neil	A&M Records	CD
	Foreign Affair	Turner, Tina	Capitol Records	Vinyl
	Hourglass	Taylor, James	CBS Records	CD
	Mariah Carey	Carey, Mariah	Columbia	CD
	Moondance	Van Morrison	Warner Records	Cassette
	Pilgrim	Clapton, Eric	Capitol Records	CD
	Revolver	Beatles, The	Capitol Records	CD
	Sweet Old World	Williams, Lucinda	A&M Records	CD
	The Dance	Fleetwood Mac	ABC Records	CD
	Time, Love & Tenderness	Bolton, Michael	Sony Music	CD
	Union	Yes	Arista	CD
				Total: 16
				Total Titles: 58
Tuesday, 11 March 2014				Page 2 of 2

Final Page

Systems Development Life Cycle

This document outlines the requirements of the Systems Development Life Cycle that is required to be completed for the database project.

A **Systems Development Life Cycle (SDLC)** is a problem-solving approach that can be used when creating computer-based solutions. The systems development life cycle consist of the following stages:

- **Problem Definition**
- **Analysis**
- **Design**
- **Development And Validation**
- **Evaluation.**

Problem Definition

(½ page)

All projects need an introduction that tells the reader what the intended aim of the developed project will be. The problem definition explains the problem/situation and outlines the intended outcomes of the system. This first phase of the systems development life cycle begins by identifying what the organisation is trying to do and how they currently achieve it.

Example:

The Client

Explain the task by describing the scenario or setting in which the problem exists. Identify the end-user and the situation that requires the database

*E.g. **The client is Bradford Library who require a database solution to accurately and efficiently keep track of books that are loaned to students.***

Outcomes

Describe four outcomes that the client requires a new system to be able to do.

These will be the actual outputs from the relational database system – that is, the reports and forms that will be generated by the new system.

- **Outcome 1: Data Entry Form**

All systems require the entry of data into a transaction table – this is the main function of the finished relational database. Produce a screen that has a form/sub-form layout.

- **Outcome 2: Multiple Criteria Report**

This outcome requires simple processing (at least two criteria) from the data in the transaction table.

- **Outcome 3: Statistical Report**

The use of grouping in Access is used to produce summary statistics such as sum, count or average from the records in a transaction table.

- **Outcome 4: Calculated Report**

This involves the creation of a new field from one or more existing fields. The ability to calculate complex calculations (such as age).

Analysis

(1- 2 pages)

In the Analysis phase, the proposed system is examined in more detail by focusing on each outcome the client requires the system to be able to do. For each outcome, the data that needs to be stored and the data processing steps required to achieve the outcome are identified. Hence all the inputs and outputs for each outcome will be listed.

In summary, the **Analysis** needs to:

- Identify the data that needs to be stored.
- Identify the processing steps to achieve the outcomes.

Example:

Break down each outcome into the data and processing steps needed to achieve the outcome. For every outcome of your database:

- State the outcome (Data Entry Form / Report)
E.g. ***A list of overdue books for each borrower***
 - Suggest the data required for the outcome (Tables and Fields)
E.g. ***SELECT Book ID, Book Title, Price FROM Books, Borrower ID, Borrower Name, Address FROM Borrowers, Date Borrowed, Due Date, Date Returned FROM Loans***
COUNT(Book ID) FROM Books
 - Identify the processing steps required for the outcome (Sorting, default value, calculated field, summary, grand summary)
E.g. ***WHERE Due Date < Date Borrowed***
GROUP_BY Borrower Id, Date Borrowed
ORDER_BY: Book Title in ascending order
-

Design

The Design stage outlines the plan for the information system, using appropriate design principles and concepts. It will propose the database and its features by discussing the entities (tables) needed based on the data identified from the outcomes.

Tables and Relationships (1 page)

- The tables needed (one paragraph per table).
- The relationships between the source tables, and why they may be one-to-many or many-to-many.
- The resolution of the many-to-many relationships between source tables with transaction tables (e.g. explain why a sales table is needed to resolve the many-to-many relationship between houses and buyers).
- Include in your discussion of each table a description of what a record in that table contains. In this way you are demonstrating a good understanding of the system you have designed.
 - For example, “**there is a record in the houses table for each house listed for sale**”, and “**one record in the buyers table for each registered buyer**”.

Table Relationship Diagram (½ page)

A table relationship diagram is a summary of the tables and their relationships in your design. This diagram may be either printed from the software in Access or produced using graphics between the tables.

- Show the links and the fields used for linking between the tables.
- On the link give the relationship between the tables and highlight the key in each table.

Data Dictionaries (½ - 1 page for each table)

These are to be of your final design and must be an accurate representation of the tables on your disk.

Marks will be given for your selection of field names, appropriate data types and field length (sizes). You should show:

- the key
- fields that have a default value to automatically supply a frequently used value e.g. `=year(now())` in the **year** field
- data entry validation checks **e.g. gender = "M" or "F"**
- input masks that are used to control how the user enters data **e.g. `PhoneNumber !(00) 0000\ 0000;0`**
- fields that have required values **e.g. a student's date of birth is required to calculate their age.**
- reducing errors at data entry **e.g. use of drop down lists for title: Mr, Mrs, Ms**

Design Efficiency and Complexity (no documentation)

This section requires no documentation. Your table relationship diagram and data dictionaries will be analysed.

A complex design is one in which there is more than the minimum of 2 source tables. This does not include simple lookup tables such as for **PostCode**, but must contain substantial data that is relevant to the situation and is needed to achieve at least one of the outcomes. For example a school database would have tables for **Students**, **Teachers** and **Classes**.

An efficient design will consider when to use keys in transaction tables, with the following to be considered in this order: double fields, triple field, or as a last resort an autonumber field. This is because the concept of a key is to uniquely identify one record. An efficient design will also consider its ease of use through the reduction of possible errors at data entry, such as auto entries, default values, pull-down lists etc.

Consideration should also be given in design to ensuring that each table has the right data. Such a process is called '**normalisation**'. Good solutions should be able to consider the following four rules:

1. Elimination of redundant data

Data that lends itself to being duplicated throughout a table should be stored only once in one table. For example, a borrower's name could be stored many times throughout the Loans table. Apart from wasting storage space, it would be possible to misspell the name of a borrower (Kethy...instead of Kathy) which will lead to inconsistencies and a lack of data integrity. By storing the name of the borrower in a separate table called Borrowers, and giving each borrower an ID (Borrower ID) only the Borrower ID needs to be stored in the Loans table, as these tables can be linked via this common field.

2. The key uniquely identifies a record in a table

As it is possible for two borrowers to have the same first and last names, the key to the Borrowers table cannot consist of these two fields. A new field, called Borrower ID needs to be created so that every student is given a different identity number and hence can be uniquely identified. For a transaction table, the key can consist of two fields – StudentID and ClassID, so that each combination of values in these fields will be unique, thereby preventing a student from being placed in the same class twice.

3. Ensuring that each field in a table relates to the key

This means that fields must be placed in the correct table. E.g. In the Book table, Borrower ID does not relate to the key Book ID, and therefore should not be stored in that table. The only exception to this principle is a field that provides a link to another table, such as a Teacher ID in the Classes table. Teacher Name should not be stored in Classes where Teacher ID already creates the link.

4. Elimination of unnecessary fields: These are fields that can be deduced from other fields in a table.

Fields must not be derived or calculated from any other field in a table. A student's age is calculated from their date of birth, hence it is the date of birth field rather than an age field that is placed in the Students table. The age is therefore defined as a calculated field in an Access query so that its value is determined each time the query runs. Another example of an unnecessary field is a field called '**returned**' that requires a **yes/no** entry. It is unnecessary when there is also another field called '**date returned**'.

Development and Validation

The **development** involves creating the database, including the tables, the queries, the forms and reports. It also includes entering in the test data and testing the database to ensure that it does what it should do. This involves documenting the tables, the query and the report (or form) and showing how the data in the final report can be found from the data in the tables.

The **validation** of your database demonstrate that your system works for the each outcome. The system is checked from all angles and involves checking for illegal transactions (e.g. two people borrowing the same book concurrently).

Validation plan (1 page)

- Produce a plan to test all features of the user interface, data input, and outcomes.
- Plan must show
 - Elements to be tested
 - Data to be used
 - Expected result
 - Sufficient evidence to confirm success

Validation of data entry form (2 pages)

Demonstrate that your system works for the *data entry screen*.

- Print the data entry form and then explain how it works.
 - Highlight the data entry fields (e.g. **Book ID**) in one colour then use the same colour to show the fields that have come from other tables by linking. Use another colour to show the linking (eg with **Borrower ID**).
 - Label all relevant fields with their special features (auto entry, drop down list etc)
- On the data dump highlight the origins of the linked data.

Validation of reports (6 – 12 pages)

Demonstrate that the reports are correct and can be achieved.

- Validate the result using the data dump. Use highlighting and comments to trace the data and/or results in the outcome with its source in the tables. Use a different colour for each table.
- Give the name of the query used to produce the outcome. Add an explanation of the processing: tables and selection criteria used, calculated fields, how it was grouped and/or sorted. (Use a labelled screen dump of the query design).
- Give the name of the report. Add explanations of the processing used and label design features such as calculated fields. State whether it was grouped/sorted or not. (Use a labelled screen dump of the report design)
- Print the report and show the data on the report including the number of records is correct as per the validation.

Your report will also be checked for sensible page layout and the inclusion of the necessary and sufficient fields.

Evaluation

(½ page)

The **evaluation** assesses the performance of the new system against the **Problem Definition**.

- Appraise the performance of your information system against the outcomes.
- Recommendations for improvements and modification if you had more time.

Presentation of the Package

- A correct life cycle layout (no appendix; put the prints as they occur in the cycle);
- A contents page with all page entries numbered (do this last and probably by hand);
- All printouts labelled and annotated.
- **You will submit your database electronically through E-learning and a hard copy of your SYSTEMS DEVELOPMENT LIFE CYCLE in the assignment box.**

Marks Scheme For Individual Project

NAME: _____

TOTAL: _____ / 140

Problem Definition (5 marks)

Low		Satisfactory		High	
0	1	2	3	4	5
How clearly does the student articulate the problem and state the desired system outcomes?					
Demonstrates limited level of understanding of the end-user's needs by: <ul style="list-style-type: none"> identifying the end-user; stating some outcomes required for the proposed system. 		Demonstrates reasonable understanding of the end-user's needs by: <ul style="list-style-type: none"> identifying the end-user and his/her situation; stating the specific outcomes required for the proposed system. 		Clearly articulates the situation and requirements of the end-user by: <ul style="list-style-type: none"> identifying the end-user and his/her situation; clearly stating all the specific outcomes and explaining how they are to be used in the system. 	

Analysis (10 Marks)

Low		Satisfactory		High	
0	1	2	3	4	5
How appropriate is the identified data?					
Demonstrates limited understanding of identification of data by presenting: <ul style="list-style-type: none"> incomplete data; irrelevant data. 		Demonstrates reasonable level of understanding of data by presenting: <ul style="list-style-type: none"> almost complete data; some irrelevant data. 		Clearly identifies appropriate data for the system by presenting: <ul style="list-style-type: none"> complete data; relevant data. 	
0	1	2	3	4	5
How appropriate are the processing steps explained to achieve a system outcome?					
Demonstrates limited understanding of processing steps by presenting: <ul style="list-style-type: none"> incomplete steps; incorrect steps; irrelevant steps. 		Demonstrates reasonable level of understanding by identifying the processing steps, of which: <ul style="list-style-type: none"> most are clear; may include a few irrelevant steps. 		Clearly identifies the processing steps, which are: <ul style="list-style-type: none"> complete; relevant; appropriately ordered. 	

Design (35 marks)

Low				Satisfactory				High			
0	1	2	3	4	5	6	7	8	9	10	
How clearly does the student discuss the tables needed based on the data identified from the outcomes?											
Demonstrates limited understanding of tables and key fields by: <ul style="list-style-type: none">stating the tables required for the solution.				Demonstrates reasonable understanding of tables and key fields by: <ul style="list-style-type: none">stating the tables and some of the fields required for the solution.				Demonstrates a high level of understanding of tables and key fields by: <ul style="list-style-type: none">clearly justifying the tables and fields required for the solution.			
How effectively does the student visually demonstrate the design of a Relational Database system?											
0		1		2		3		4		5	
Produces a design that is of limited effectiveness by: <ul style="list-style-type: none">providing an incomplete diagram.				Produces a design that is of reasonable effectiveness by: <ul style="list-style-type: none">providing a simple visual diagram.				Produces a design that is of high effectiveness by: <ul style="list-style-type: none">providing a detailed and complete visual diagram.			
How effectively has the student displayed the table/field properties (data dictionaries)?											
0	1	2	3	4	5	6	7	8	9	10	
Demonstrates a limited level of skill and understanding of design by producing a design that: <ul style="list-style-type: none">considers data type;is incomplete.				Demonstrates a reasonable level of skill and understanding of design by producing a design that: <ul style="list-style-type: none">considers data type and some properties;is inaccurate.				Demonstrates a high level of skill and understanding of design by producing a design that: <ul style="list-style-type: none">considers data type and detailed field properties.is accurate;			

How complex and efficient is the design of the proposed system?										
0	1	2	3	4	5	6	7	8	9	10
Produces a design with a low level of efficiency because it: <ul style="list-style-type: none"> is incomplete. 				Produces a design that is reasonably efficient because it: <ul style="list-style-type: none"> uses only two source tables includes irrelevant data; is partially complete; some fields in the table do not relate to the key 			Produces a design that is highly efficient because it: <ul style="list-style-type: none"> uses more than two source tables considers data that is relevant to all outcomes; (no unnecessary fields) fields in each table relate to the key all non-key fields are dependant on the key 			

Development and Validation (70 marks)

Low					Satisfactory					High																				
How user-friendly is the developed system?																														
User interface																														
0	1	2	3	4	5	6	7	8	9	10																				
Applies a limited level of knowledge, skill and understanding to the user interface, as evidenced by: <ul style="list-style-type: none">layout and appearance that does not adhere to design principles;					Applies a reasonable level of knowledge, skill and understanding to the user interface, as evidenced by: <ul style="list-style-type: none">appropriate layout and appearance;successful navigation.					Applies a high level of knowledge, skill and understanding to the user interface, as evidenced by: <ul style="list-style-type: none">effective layout and appropriate appearance;ease of use and navigation; (switchboard)successful navigation.																				
How effectively does the student apply appropriate knowledge, skills, and processes to achieve an outcome or solve a problem?																														
Use of data entry form/layout to enter new source and transaction data																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15															
Applies a limited level of skill and knowledge of data input by: <ul style="list-style-type: none">using inappropriate layout and appearance;not allowing data to be processed.					Applies a reasonable level of skill and knowledge of data input by: <ul style="list-style-type: none">using appropriate layout and appearance;successfully processing data. – linking works					Applies a high level of skill and knowledge of data input by: <ul style="list-style-type: none">using effective layout (drop down lists, checkboxes, subform) and appropriate appearance;successfully processing all data – linking workssystem works from all angles – no illegal transactions																				
Outcomes (Reports)																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Applies a limited level of skill and knowledge by: <ul style="list-style-type: none">achieving some outcomes;using inappropriate layout and appearance;										Applies a reasonable level of skill and knowledge by: <ul style="list-style-type: none">achieving most outcomes;using appropriate layout and appearance;										Applies a high level of skill and knowledge by: <ul style="list-style-type: none">achieving all outcomes;using effective layout and appropriate appearance; (easy to read)										

Development and Validation (cont'd)

How accurate and thorough is the validation plan?										
0		1		2		3		4		5
Produces a validation plan that is of limited accuracy and reliability because it includes: <ul style="list-style-type: none">few features of the user interface, data input and outcomes to be tested;insufficient data to test the feature.				Produces a validation plan that is of reasonable accuracy and reliability because it includes: <ul style="list-style-type: none">some features of the user interface, data input, and outcomes to be tested;sufficient proposed data to test the featuresufficient proposed evidence to confirm success.				Produces a validation plan that is of high accuracy and reliability because it includes: <ul style="list-style-type: none">all features of the user interface, data input, and outcomes to be tested;sufficient proposed data to thoroughly test the feature;sufficient proposed evidence to confirm success.		
Low				Satisfactory				High		
How thoroughly does the student test the elements of the solution?										
0	1	2	3	4	5	6	7	8	9	10
Provides little or no evidence of testing of the system.				Demonstrates a reasonable level of testing by providing: <ul style="list-style-type: none">documentation shows the student has tested some of the system according to the validation plan;verification of the results.				Demonstrates a high level of testing by providing: <ul style="list-style-type: none">documentation that shows the student has tested the system fully according to the validation plan;verification of the results.		

Evaluation (10 marks)

How effectively does the student discuss the performance of your information system against the outcomes.					
0	1	2	3	4	5
Demonstrates a limited level of understanding of evaluation by: <ul style="list-style-type: none">making limited comment on how effective the solution is in meeting the problem definition.		Demonstrates a reasonable level of understanding of evaluation by: <ul style="list-style-type: none">stating how effective the solution is in meeting the problem definition.		Demonstrates a high level of evaluation by: <ul style="list-style-type: none">discussing the effectiveness of the solution and identifying any areas that may need further attention.	
How appropriate are the student's recommendations for improvements and modification?					
0	1	2	3	4	5
Produces recommendations that are inappropriate or insufficient by: <ul style="list-style-type: none">making few recommendations;making invalid recommendations for improvement.		Produces recommendations that are reasonably appropriate by: <ul style="list-style-type: none">making appropriate recommendations on improvements that could have been applied to the developed system.		Produces recommendations that are highly appropriate by: <ul style="list-style-type: none">making and justifying appropriate recommendations on improvements that could have been applied to the developed system.	

Presentation of the Project (10 marks)

<i>Low</i>				<i>Satisfactory</i>				<i>High</i>		
1	2	3	4	5	6	7	8	9	10	
How effectively does the student document the stages of the Systems Development Life Cycle.										
Produces documentation that is of limited effectiveness by providing: <ul style="list-style-type: none"> an incorrect life cycle layout. some sections with correct headings. sub standard use of grammar minimal use of quality word-processing. 				Produces documentation that is of reasonable effectiveness by providing: <ul style="list-style-type: none"> correct life cycle layout. all sections with correct headings. satisfactory use of grammar reasonable use of quality word-processing. 				Produces documentation that is of high effectiveness by providing: <ul style="list-style-type: none"> Correct life cycle layout. All sections with correct headings. A contents page with all pages numbered. Good use of grammar, and high-quality word-processing. 		

Definitions

Reference: Unit J infoweb – W. J. Gilmore

<http://www.devshed.com/c/a/MySQL/An-Introduction-to-Database-Normalization/>

Entity

The word 'entity' as it relates to databases can be defined as the general name for the information that is to be stored within a single table. For example, if you were interested in storing information about the school's students, then 'student' would be the entity. The student entity would likely be composed of several pieces of information, for example: student identification number, name, and email address. These pieces of information are better known as attributes.

Primary key

A primary key uniquely identifies a row of data found within a table. Referring to the school system, the student identification number would be the primary key for the student table since an ID would uniquely identify each student.

Relationship

Define how entities in different tables relate to each other. There are three types of data relationships that you should be aware of:

- **one-to-one (1:1)** - A one-to-one relationship signifies that each instance of a given entity relates to exactly one instance of another entity. E.g. each student would have exactly one grade record, and each grade record would be specific to one student.
- **one-to-many (1:M)** - A one-to-many relationship signifies that each instance of a given entity relates to one or more instances of another entity. E.g. one teacher entity could be found teaching several classes, and each class could in turn be mapped to one teacher.
- **many-to-many (M:M)** - A many-to-many relationship signifies that many instances of a given entity relate to many instances of another entity. E.g. a schedule could be comprised of many classes, and a class could be found within many schedules.

Foreign Key

A foreign key forms the basis of a 1:M relationship between two tables. The foreign key can be found within the M table, and maps to the primary key found in the 1 table. To illustrate, the primary key in the teacher table (probably a unique identification number) would be introduced as the foreign key within the classes entity, since it would be necessary to map a particular professor to several classes.

Entity-Relationship Diagram (ERD)

A graphical representation of the database structure. These diagrams are immensely useful towards attaining a better understanding of the dynamics of the various database relationships.

Transaction Table

Resolves the many-to-many relationship between two tables. As a M:M relationship cannot be implemented, a transaction table is formed to join the two tables. Other information is also stored in this table. For example, in the schedule entity (table), the time and room number of the class would also be stored, along with the student id and the class id.

Data Redundancy

is a repetition of data or the storage of data that can be produced or calculated with a query. Data that is repeated wastes space and over time can cause errors in the database as some parts are updated and others are not. Data that can be calculated, such as a person's age, wastes space and quickly becomes wrong and out of date.

Foundation Studies (CIS)

Topic 9: Programming Using Javascript



```
<!DOCTYPE html>
<html xmlns='
<head>
  <meta ht
```

Objectives

1. Define a computer program
2. Describe the methods of translating a program language into machine code
3. Create a HTML page using Javascript
4. Describe an algorithm
5. Describe psuedocode
6. List the basic computer processing operations
7. Create algorithms using control structures: sequence, selection, repetition.
8. Select appropriate data to deskcheck algorithms
9. Create appropriate loops to repeat a set of statements
10. Create loops to validate data
11. Define variables and data types
12. Perform calculations and comparisons on data
13. Define a function
14. Create forms and elements to accept user input
15. Create text fields, radio buttons, checkboxes and buttons

Computer Programs

A **computer program** is a list of instructions that tells the computer what to do. The computer follows these instructions, one by one, until it reaches the end of the program.

Computers don't understand English. They only understand **machine code** which is a series of 1's and 0's. Therefore, we require programming languages such as **Javascript** to allow people to write programs in English like languages that a web browser will interpret and change into machine code, which the computer can understand.

To create a web page programmers use the language of HTML. They may use a software application such as DreamWeaver as a way to create and organise the coding.

Programming languages (e.g. Javascript, C++, Python) all need to be **translated** into machine code from the language that they were created in before they can be executed and run.

There are two methods of translating a programming language into machine code. These are via a **compiler** or an **interpreter**.

- A **compiler** runs through all the lines of code and translates it into a machine code file (.com or .exe for example), which can then be executed or run.
- An **interpreter** on the other hand translates each line of code as it runs. Your web browser translates each line of HTML code this way.

Javascript was created to allow web developers to embed executable code on their webpages, so that they could make their webpages interactive, or perform simple tasks. i.e. a user can click on buttons you create to select various items they wish to purchase and your code will be able to calculate a total price due.

There are 2 main sections in this document. They are:

ALGORITHMS & JAVASCRIPT

Algorithms

An **algorithm** is a series of steps written in sequence to define the solution to a problem

- A recipe for baking cake is an algorithm!
- Algorithms written in a computer language are called computer programs.
- They are used by programmers to ensure that the program runs exactly as intended and that it can take all eventualities into consideration.
- An algorithm is a list of precise steps, so the order of computation will always be critical to the correct functioning of a program.
- Algorithms contain the essential logic of a program and for this reason they are constructed and tested thoroughly before any coding is written.

Pseudo-Code

Pseudo-code is an outline of an algorithm written in a way that is between English and a programming language. It can easily be converted into real programming statements.

- It is the method for writing a program without using the correct words (syntax) of the programming language.
- The purpose of using pseudo-code is that it is easier for humans to understand than programming language code.
- It has no fixed rules but it must make sense.
- You can write pseudo-code without even knowing what programming language you will use for the final implementation.

Control Structures

Algorithms will include all the steps, including the input of the data and the output of the result. All algorithms, and all programs, are written with the use of just 3 things;

- **Sequence**
- **Selection**
- **Repetition**

It is the order of the statements in an algorithm that matters.

Sequence

All instructions are executed one after another. For example, they might receive data, perform calculations and assign the results to variables.

Selection

This is used to compare two pieces of data, and use the result to select an action or a sequence of actions.

Repetition

Allows the same set of instructions, a sequence, to be executed a number of times in succession.

Deskchecks

- When an algorithm is written, it is important to do a desk-check to ensure that the result you get is what was expected.
- Deskchecking involves tracing through the logic of the algorithm with some chosen data.
- This is usually in the form of a table, with each variable that is used being written once along the top row, and the steps, the different lines, of the algorithm being written down the left hand column. From then on, the cells of the table are filled in as each line in the algorithm dictates.

Order of Steps in an Algorithm:

Suppose we had an algorithm that looked like this:

```
A = B + C
D = D + A
DISPLAY D
```

If the variables A, B, C, D contain the following values: A=10, B=20, C=5, D=1, a desk-check for the following algorithm is shown below.

A	B	C	D		OUTPUT
10	20	5	1		
25					
			26		
					26

The final result displayed will be 26

At the end of the algorithm, the variables A, B, C and D contain the following values: A=25, B=20, C=5, D=26

However, if we keep the same start values for variables A, B, C, D, but change the order of the statements in the algorithm the desk-check will appear as follows:

Desk Check:

A	B	C	D	OUTPUT
10	20	5	1	
			11	
25				
				11

The final result displayed will be 11.

Sequence Control Structure

The **SEQUENCE control structure** means that the steps are performed one after another. These instructions must be executed as they appear, in a top-to-bottom fashion.

Example 1

Create an algorithm that will calculate the final price for a customer who is purchasing a number of chairs, worth \$55 each. Display the final price.

Note: Variables to hold the different values need to be defined. These might be: numChairs, chairPrice and finalPrice

```
START
    var numChairs, chairPrice, finalPrice
    input numChairs
    chairPrice = 55
    finalPrice = chairPrice * numChairs
    Display "The price owing is", finalPrice
END
```

To check if this algorithm works we need to do a desk check.

A desk-check for that input would be:

NumChairs	ChairPrice	FinalPrice	OUTPUT
2			
	55		
		110	
			The price owing is 110

Example 2

Create an algorithm that will calculate the area of a rectangle. Display the area.

The above problem in pseudo-code:

```
START
    var length, width, area
    get length, width
    area = length * width
    display area
END
```

END

Complete the desk check for the following data:

Desk Check: 5, 10

length	width	area	OUTPUT

Example 3

Create an algorithm that will update an account balance with an interest payment. The interest payment is 8%. Display the final amount.

The above problem in pseudo-code:

START

```
var accountBalance, interestRate, accountInterest
input accountBalance
interestRate = 0.08
accountInterest = accountBalance * interestRate
accountBalance = accountBalance + accountInterest
display accountBalance
```

END

Complete the desk check for the following data:

Desk Check: 100

accountBalance	interestRate	accountInterest	OUTPUT

Example 4

Create an algorithm to accept the quantity of wines and a unit price, then display the cost of purchasing the wines.

START

```
var quantity, price, cost
input quantity, price
cost = quantity * price
display cost
```

END

Complete the desk check for the following data:

Desk Check: 3, 15

quantity	price	cost	OUTPUT

Exercise 1 – Sequence

1. Create an algorithm and deskcheck that requests a number in inches and converts it into centimetres.
(Note: 1 inch = 2.54cm)

2. Create an algorithm and deskcheck that requests 3 test results and calculates the average.

3. Create an algorithm and deskcheck that asks the name and age of the user in years and outputs the age in months as follows: *Hello <<name>>. Are you really <<age> months old.*

Selection Control Structure

The **SELECTION control structure** means that steps are performed only under certain conditions.

- The instructions where two pieces of data are compared and the result is used to select an action or a sequence of actions between two or more alternatives.
- This control structure represents the decision ability of the computer.

IF (condition is true) THEN

OR

IF (condition is true) THEN

step(s) if condition is true

step(s) if condition is true

ENDIF

ELSE

step(s) if condition is false

ENDIF

```
var num
num=12
if num=12 then
    display "It was true"
endif
```

```
var num
num=12
if num=12 then
    display "It was true"
else
    display "It was false"
endif
```

Conditional Operators

Operator	Comparative Test
==	Equality
!=	Inequality
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Example 5

Create an algorithm that will display the service charge for accounts. Accounts that have a balance less than \$400 get charged a service charge of \$3, and all other accounts get charged \$5.

Test data for this algorithm should be:

Test Data	Expected Result	Reason
100.....	3	this data is obviously less than 400
400.....	5	this data is known as cusp data.
500.....	5	this data is obviously greater than 400

A **cusp** is the point at which the choice can change from one value to another. In this case the service charge changes from \$3 to \$5. It is VERY important that these cusp points are checked

START

```

var accountBalance, serviceCharge
input accountBalance
if accountBalance < 400 then
    serviceCharge = 3
else
    serviceCharge = 5
endif
display "the service charge on this account is: $", serviceCharge

```

END

There is a deliberate indentation here inside the "Then" and "Else". This is to help with readability of code. You MUST use indentation in all your code

Desk-Check: If the sample input data is 100

AccountBalance	ServiceCharge	OUTPUT
100		
	3	
		The service charge on this account is \$3

Desk-Check: sample input data is 400

AccountBalance	ServiceCharge	OUTPUT
		The service charge on this account is \$

Desk-Check: data is 500

AccountBalance	ServiceCharge	OUTPUT
		The service charge on this account is \$

Example 6

The membership category for a health club depends upon the member's age. Members over the age of 55 are given a category of "senior". Members over 21 up and to 55 are "adults", and 21 or younger are considered to be "junior" members. Display the membership category of any member.

The above problem in pseudo-code:

```
START
    var age, category
    input age
    if age >55 then
        category = "senior"
    elseif age <=21 Then
        category = "junior"
    else
        category = "adult"
    endif
    display "The membership category is : ", category
END
```

Desk-Check: sample input data is 18

Age	Category	OUTPUT
18		
	junior	
		The membership category is : junior

Desk-check: data is 21

Age	Category	OUTPUT
		The membership category is :

Desk-check: 45

Age	Category	OUTPUT
		The membership category is :

Desk-check: 55

Age	Category	OUTPUT
		The membership category is :

Desk-check: 60

Age	Category	OUTPUT
		The membership category is :

Example 7

Try the above problem (algorithm example 3) with a different algorithm with the same test data

START

```

var age, category
Input age
If age >0 Then
    Let category = "junior"
Elseif age >21
    Let category = "adult"
Else
    Let category = "senior"
End if
Display category

```

END

Desk check: 18

Age	Category	OUTPUT
		The membership category is :

Desk-check: 60

Age	Category	OUTPUT
		The membership category is :

Task 1

Explain what the algorithm is doing for "Example 7".

Example 8

Create an algorithm to update an account balance with an interest payment. The interest payment is calculated at a rate of 8% pa for balances less than 10,000 and at a rate of 10% for balances of 10,000 or more. Display the final amount.

START

```
var accountBalance, interestRate, accountInterest
Input accountBalance
If accountBalance < 10000
Then
    Let interestRate = 0.08
Else
    Let interestRate = 0.1
Endif
Let accountInterest = accountBalance * interestRate
Let accountBalance = accountBalance + accountInterest
Display accountBalance
```

END

Desk Check: 20000

AccountBalance	InterestRate	AccountInterest	OUTPUT

Desk Check: 10000

AccountBalance	InterestRate	AccountInterest	OUTPUT

Desk Check: 500

AccountBalance	InterestRate	AccountInterest	OUTPUT

Exercise 2 – Selection

1. Create an algorithm that asks the age of a child (in years) and decides whether the child is old enough to attend school.

(**Note:** child must be 5 years old to attend school)

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and extend across the width of the page. There are no margins, text, or other markings on the paper.

2. Write an algorithm that prompts the age of a person.
 - If age is less than or equal to 12, the program displays “You are a child”
 - If age is between 13 and 19 (inclusive), display “You are a teenager”
 - If age is greater than or equal to 20 output ‘You are an adult”

[illegible]

3. Create an algorithm and the deskchecks prompts a student for three test results, each out of 20 marks. The overall grade is to be displayed.
 - an A is 51 or higher
 - a B is 42 or higher
 - a C is 33 or highe
 - a D is 32 or less

[illegible]

Repetition Control Structure (Loops/Iteration)

REPETITION control structure involves the process of repeating a set of instructions over and over (this is often called looping).

- Loops will continue to execute as long as a condition is satisfied or True
- It is essential to determine how the loop will stop, and to make sure that this actually happens.
- The condition is tested every time the loop is processed, to determine whether or not to continue with the loop, or to end it. There are two common methods used to terminate the loop.

1. If the number of iterations is known beforehand.

A **counter** can be used and it can be tested each time, to see whether the loop has been executed for the correct number of times. You must Increase (**increment**) or decrease (**decrement**) the counter each time through the loop.

E.g. if a company is calculating the pay for its 20 employees, then the algorithm will loop 20 times, and the counter can count from 1 to 20 to determine when the loop is to terminate.

2. If the number of iterations (times of repeating) is not known.

A a "**terminating condition**" is input at the end of the list of data. The loop repeats until the terminating condition is input.

E.g. if the number of employees changes from week to week, the pay algorithm may have to loop any number of times. For example, assuming that employee ID is always positive, the loop could end when an employee ID of -9 is input.

Types of Loops

There are two types of loops:

- **Pre-test**
- **Post-test**

The Pre-Test Loop

The **pre-test** loop will test for the condition **before** entering the loop. This means that the loop may not be entered at all if the test isn't satisfied.

The Post-Test Loop

The **post-test** loop tests for the condition **at the end** of the loop. This means that the loop is always executed at least once.

There is usually a number of test data, so that the loop can continue until the terminating condition is reached. Desk checks for loops can sometimes continue for many pages!

Example 9

Write an algorithm that displays the numbers from 1 to a typed number.

START

```
var count, num
```

```
input count
```

```
num = 1
```

```
while num <= count
```

```
    display num
```

```
    num= num + 1
```

```
end while
```

END

num is initialized before the WHILE condition is executed

As long as **num** is less than or equal to count, the steps will be repeated. When **num** is not less than count, the WHILE condition becomes false and the repetition stops.

The initialising (**num = 1**) and the incrementing/decrementing (**num = num + 1**) of the variable tested (i.e. **num**) is an important feature of the WHILE construct.

A desk check for this is as follows: (**test data: 5**)

count	num	OUTPUT
5		
	1	
		1
	2	
		2
	3	
		3
	4	
		4
	5	
		5
	6	

Example 10

The While Loop – a pre-test loop.

```

1. var payRate, numEmployees, numHours, pay
2. Let payRate = 10
3. Let numEmployees = 5
4. While numEmployees > 0
5.   Input numHours
6.   Let pay = numHours * payRate
7.   Display "the pay for employee is: $", pay
8.   numEmployees = numEmployees - 1
10. End While

```

The **While** loop condition is tested and the loop is only entered if the condition is true (i.e. **numEmployees > 0**)

The Loop controller, **numEmployees**, is **decremented** (made smaller) to make sure the loop will terminate by reaching 0 at some point.

A desk check for this is as follows:

	payRate	numEmployees	numHours	pay	OUTPUT
3	10				
4		5			
6			15		
7				150	
8					the pay for employee is: \$150
9, 5		4			
6			20		
7				200	
8					the pay for employee is: \$200
9, 5		3			
6			10		
7				100	
8					the pay for employee is: \$100
9, 5		2			
6			8		
7				80	
8					the pay for employee is: \$80
9, 5		1			
6			40		
7				400	
8					the pay for employee is: \$400
9, 5		0			

Example 11

The Do-While Loop – A Post-Test Loop.

```

1. var payRate, numEmployees, numHours, pay
2. Let payRate = 10
3. Let numEmployees = 5
4. Do
5.   Input numHours
6.   Let pay = numHours * payRate
7.   Display "the pay for employee is: $", pay
8.   numEmployees = numEmployees - 1
10. While numEmployees > 0

```

The loop is entered first and only after the loop has run once, is the condition tested. Only then, if the condition is true, does the loop get entered again.

Try a desk check for this, using the same test data: 15, 20, 10, 8, 40

	payRate	numEmployees	numHours	pay	OUTPUT
3	10				
4		5			
6			15		
7				150	
8					the pay for employee is: \$
9		4			
6			20		
7					
8					the pay for employee is: \$
9		3			
6			10		
7					
8					the pay for employee is: \$
9		2			
			8		
					the pay for employee is: \$
		1			
			40		
					the pay for employee is: \$
		0			

Example 13

Comparison Of The Two Types Of Loops.

Compare the following two algorithms, which calculate the average of a list of numbers greater than zero. It keeps inputting numbers until a zero is input.

Start

```
var count, total, num, avg
count = 0
total = 0
Input num
While num <> 0
    Let total = total + num
    Let count = count + 1
    Input num
End While
Let avg = total/count
Display "the average is:", avg
```

End

Test Data: 2,4,1,1,0

Start

```
var count, total, num, avg
count = 0
total = 0
Do
    Input num
    Let total = total + num
    let count = count + 1
While num<> 0
Let avg = total/count
Display "the average is:", avg
End
```

Test Data: 2,4,1,1,0

Use the tables on the next page to do the deskschecks to compare your result.

Exercise 3 - Repetition

1. These loops give different results. Which one is accurate? Why?

2. What happens in both cases if the test data is: 0,2,4,0 ? (Draw up a desk check for each to determine this.)

3. How are the loops terminated?

4. What do you think a logic error is?

5. Write an algorithm and a deskcheck for a program that will write your name three times.

[illegible]

Data Validation

Loops are frequently used to ensure that the data being input is valid. This could be used to ensure that a number is within a certain range (ie >0, <100, <0...), or that certain words are input (such as a correct password).

Examples: Ensuring that the password entered is HAZEL.

A – Using An IF Test

```
var password
Input password
IF password <> "HAZEL"
Then
    Display "invalid password, reenter"
    Input password
End if
...
...(continue with processing)
...
```

B – Using The WHILE Loop

```
var password
Input password
While password <> "HAZEL"
    Display "invalid password, reenter"
    Input password
End While
...
...(continue with processing)
```

C – Using The DO-WHILE Loop Variation 1

```
var password
Input password
Do
    Display "invalid password, reenter"
    Input password
While password <> "HAZEL"
...
...(continue with processing)
```

D – Using The DO-WHILE Loop Variation 2

```
var password
Do
    Input password
While password <> "HAZEL"
...
...(continue with processing)
...
```

Question: Which of the algorithms on the previous page is the better method to use in this case? Why?

More Than One Test Condition:

If you want to test more than one condition, you can join the conditions with either an **AND** or an **OR** statement, depending upon the logic.

AND – **both** conditions must be TRUE for the whole statement to be TRUE

OR – **either** condition can be TRUE for the whole statement to be TRUE

Example 1: To ensure that a number is greater than 0 and less than or equal to 100:

```
Input num
While (num <=0) OR (num >100)
    Display "number out of range"
    Input num
End While
...
...(continue with statements)
...
```

Example 2: To ensure that a password entered is either "HAZEL" or "ZIGGY":

```
Input password
While (password <> "HAZEL") AND (password <> "ZIGGY")
    Display "password is invalid - try again"
    Input password
End While
...
...(continue with statements)
```

Basic Computer Processing Operations

There are six basic computer processing operations and the following are examples showing various pseudo-code that might be used in each:

1. Receiving Data

A computer is required to receive data or input from a particular source (i.e. a user).

Keywords used: INPUT, GET, READ

```
Input student_name  
  
Read number_of_chairs  
  
Get student_id_number
```

2. Output of Information

A computer may be required to supply information or output to a device (i.e. screen, printer)

Keywords used: DISPLAY, PRINT, OUTPUT, WRITE, PUT

```
Display final_price  
  
Print "The amount owing is", owing  
  
Output totaTax
```

3. Performing calculations

Most computer programs require the computer to perform some sort of mathematical calculations or formula. The equal symbol "=", has been used to indicate the assignment of a value as a result of some processing.

*Symbols used: +, -, *, / ()*

```
totalPrice = chairPrice * numChairs  
  
total = total + number  
  
add number to (total + sum)
```

4. Assigning a value to a variable

Three cases where you may write pseudocode to assign a value to a piece of data:

1. To give data a starting value
2. To assign a value as a result of a calculation
3. To keep a piece of data for later use

```
count = 0  
  
set total to 0  
  
balance = openingBalance - closingBalance
```

5. Comparing two pieces of data

A computer program has the ability to compare two pieces of information and then as a result of a comparison, select one of two alternative actions.

Keywords used: IF, THEN, ELSE, ENDIF

```
If discount = 0
Then
    finalPrice = price
Else
    finalPrice = (price* discount) + price
End if
```

6. Repeating a group of actions

A computer program has the ability to repeat a sequence of steps over and over again.

Keywords used: WHILE, DO, DOWHILE, ENDDO

```
Let numberOfItems = 0
Let totalPrice = 0
While numberOfItems < 5
    Input price
    Let totalPrice = totalPrice + price
    Let numberOfItems = numberOfItems + 1
End while
Display totalPrice
```

Exercise 4 - Algorithms

Write the pseudo-code algorithm for each of the following questions. Also complete a desk check for each question to check that the algorithm is doing what it should be doing. Make sure this exercise is included in your **PORTFOLIO**.

1. Write an algorithm that will calculate the price of a purchase of some office equipment. The buyer wants to buy some desks @ \$205 each and some chairs @ \$65 each. The number of desks and chairs to purchase will be input. Display the total price to pay.
2. Write an algorithm that will calculate the price of a purchase of computer equipment. The buyer wants to buy some laptops @ \$2205 each and some printers @ \$365 each. The shop has advertised a 10% discount on all prices. The number of laptops and printers to purchase will be input. Display the total price to pay.
3. Write an algorithm to accept a quantity of wines and a unit price, then display the cost of purchasing the wines. A discount of 15% is given when a dozen (or more) wines are bought.
4. Write an algorithm that accepts the hours worked and rate of pay, then displays the gross pay, given that each hour over 38 hours worked is paid at one and half time the rate.
5. Calculate the pay due for 6 employees. The pay rate and the number of hours worked is input and the pay is calculated and displayed. Overtime is payable at the rate of 1.5 for all hours worked above 38. Also display a message "overtime calculated" if the employee has worked overtime.
6. Calculate the pay due for 6 employees. The pay rate is the same for all employees and is input once. The number of hours worked is input for each employee and the pay is calculated and displayed. Overtime is payable at the rate of 1.5 for all hours worked above 38. Also display a message "overtime calculated" if the employee has worked overtime.

Ensure that the number of hours worked is between 5 and 50, inclusive.

7. Change Algorithm exercise 4 above so that when all the pays have been calculated, the total amount (ie the sum) of all the pays is displayed.

Introduction to Javascript

JavaScript is a special interpreted programming language whose interpreter is built into a web browser. It allows you to create dynamic web pages that allow interaction with the user. One of the main reasons for using a scripting language is to move as much processing as possible from the server to the client's computer.

The main difference between a scripting language and a general purpose programming language is that the script must be opened within a browser for the script to be interpreted.

Using the SCRIPT Tag

You can embed JavaScript in a HTML document as statements and functions within a **<SCRIPT>** tag. This is an extension to HTML that can enclose any number of JavaScript statements as shown here:

```
<SCRIPT>
    JavaScript statements...
</SCRIPT>
```

A document can have multiple SCRIPT tags, and each can enclose any number of JavaScript statements.

Embedding JavaScript in HTML

In order to include JavaScript in a web document, a script block must first be defined within the HTML code. This alerts the browser that the code that follows needs to be interpreted as a script.

Unlike HTML, JavaScript is case sensitive!!!!!!.

Specifying the JavaScript Version

The TYPE attribute lets the browser know that the script being used is JavaScript.

```
<SCRIPT TYPE="text/javascript">
    JavaScript statements...
</SCRIPT>
```

Output – The Alert Function

JavaScript provides a function that allows you to output a message or the contents of a variable (more on variables later). It will display in an alert box.

To display a message "Hello, This is a message", write:

```
alert("Hello, This is a message");
```

To display the contents of the variable pay, write:

```
alert(pay);
```

To display a message and variable, write:

```
alert("the pay is " + pay);
```

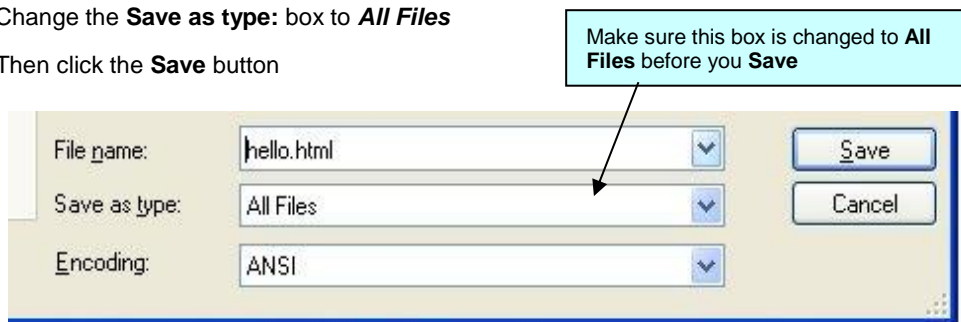
Note the use of the semicolon (;) All JavaScript statements must end with one.

Exercise 5 - Programming

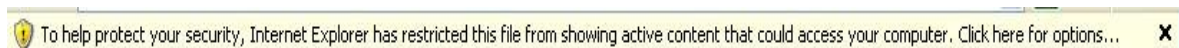
1. Open **Notepad** and copy the following text.

```
<HTML>
<HEAD>
<title>output</title>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    alert ("Hello world");
</SCRIPT>
<P> That's all folks
</BODY>
</HTML>
```

2. To save the page click **File – Save As...**
3. Type **hello.html** in the **File name:** box
4. Change the **Save as type:** box to **All Files**
5. Then click the **Save** button



6. To run the script, open **Internet Explorer** and drag the file onto the browser. If the script is blocked click in the security message at the top of the browser and "**Allow blocked content**" the script to be run.



Variables

- A **variable** is name that is given to a space in memory, where we can store a word, number or phrase.
- To create a variable you must first **declare** it: e.g. `var age`
- You can then place or **assign** data in this variable as follows: e.g. `age = 21`
- Variable names should start with a letter (can start with underscore `_`), should not contain punctuation and should not be a **reserved word**.
- Examples of variable names:

```
Counter  
firstWord  
pageLoaded  
pageNumber  
thisVariable
```

Reserved Words:

```
Abstract, Boolean, break, byte, case, catch, char, class, const, continue,  
default, delete, do, double, else, extends, false, final, finally, float,  
for, function, goto, instanceof, implements, import, in, int, interface, long,  
native, new, null, package, private, protected, public, return, short, static,  
super, switch, synchronized, this, throw, throws, transient, true, try,  
typeof, var, void, while, with
```

Exercise 6 - Variables

1. Change your previous example (**hello.html**) to use a variable (**message**) as follows:

```
<HTML>  
<HEAD>  
<TITLE>Example 2</TITLE>  
</HEAD>  
<BODY>  
<SCRIPT TYPE="text/javascript">  
    var message = "Hello world";  
    alert( message);  
</SCRIPT>  
<P> That's all folks  
</BODY>  
</HTML>
```

← You must declare your variables. To do this in JavaScript you use the Reserved word **var** to declare the variable called **message**

2. Save the file as **hello2.html** and run it.
3. What happens if we type `alert("message")` instead of `alert(message)` in the above script?

-
-
4. Display an alert with your name.
 5. Display an alert with your age.

Exercise 7 – Assigning Values to Variables

1. Enter the following:

```
<HTML>
<HEAD>
<TITLE>Pay 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var pay = 56;
    alert( "Your pay is $" + pay);
</SCRIPT>
<P> That's all folks
</BODY>
</HTML>
```

Notice that the code is indented. It is important to indent your code to make it more readable.

2. Enter the following:

```
<HTML>
<HEAD>
<TITLE>Pay 2</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var numHours = 40;
    var payRate = 25;
    var pay = numHours * payRate;
    alert( "Your pay is $" + pay);
</SCRIPT>
<P> That's all folks
</BODY>
</HTML>
```

Data Types

A value, the data assigned to a variable, may consist of any sort of data. However, JavaScript considers data to fall into several possible types. Depending on the type of data, certain operations may or may not be able to be performed on the values. For example, you cannot arithmetically multiply two string values. Variables can be of these types:

Number	3 or 7.987, Integer and floating-point numbers. Integers can be positive, 0, or negative A floating-point number can contain either a decimal point, an "e" (uppercase or lowercase), which is used to represent "ten to the power of" in scientific notation, or both.
Boolean	True or False. The possible Boolean values are true and false. These are special values, and are not usable as 1 and 0. In a comparison, any expression that evaluates to 0 is taken to be false, and any statement that evaluates to a number other than 0 is taken to be true.
String	"Hello World !" Strings are delineated by single or double quotation marks. (Use single quotes to type strings that contain quotation marks.)

JavaScript is known as a loosely typed language -- you do not have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution. You may simply assign any type of data to any variable. The only time data typing matters is when you need to perform operations on the data. Certain operators behave differently depending on the type of data being deal with. For example, consider the + operator:

"total " + "price"	yields	"total price" (string concatenation)
"5" + "10"	yields	"510" (string concatenation)
5 + 10	yields	15 (arithmetic sum)

Exercise 8 – Types of Variables

1. Use the **typeof** keyword to show the type of each of the variables a, b and c:

```
<HTML>
<HEAD>
<TITLE> Typeof 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var a = 0.08;
    var b = "My name is John";
    var c = false;
    alert( typeof a);
    alert( typeof b);
    alert( typeof c);
</SCRIPT>
<P> That's all folks
</BODY>
</HTML>
```

Input - The Prompt box

Prompt box allows the user to enter in the data, instead of it being pre-programmed into the code. Its format is:

```
variable = prompt("string");
```

- or -

```
variable = prompt("string", "default value");
```

where the default value is an optional extra

The string will be displayed in an input box, which will remain on the screen so that data can be entered and the OK button pressed. The value will be placed into the variable. It will be stored as type STRING.

Exercise 9 - Input

1. Enter the following:

```
<HTML>
<HEAD>
<TITLE>Prompt 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var message;
    message = prompt("enter your name");
    alert( message);
</SCRIPT>
<P> That's all folks
</BODY>
</HTML>
```

You must declare your variables. To do this in JavaScript you use the Reserved word **var** to declare the variable called **message**

2. Change the above script to replace the line

```
    alert(message);
with
    alert( "hello " + message);
```

and

```
    <P> That's all folks
with
    <P> That's all <SCRIPT>document.write(message);</SCRIPT>
```

3. Did you press the space key after typing in the word "hello " and "That's all " in the above script? What happens if you don't do this?

Performing Calculations

Arithmetical Operators

Operator	Operation
+	Addition and also concatenates strings
-	Subtraction
*	Multiplication
/	Division
++	Increment
--	Decrement

Examples:

```
var addNum = 20 + 30;
var addStr = "I love" + "JavaScript";
var mul = 8 * 5;
var inc = 7; inc = ++inc;
```

Exercise 10 – Performing Calculations

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>add</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var sum1, sum2, total;
    sum1 = 3 + 4;
    sum2 = 4 + 5 ;
    total = sum1 + sum2;
    alert(total);
</SCRIPT>
<P> All done.
</BODY>
</HTML>
```

Remember to declare your variables

2. Try it out:

```
<HTML>
<HEAD>
<TITLE>discount</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var discount, price, total;
    discount = 0.1;
    price = 56;
    total =price- (price * discount);
    alert("The final price with discount of " + discount + " is " + total);
</SCRIPT>
<P> All done.
</BODY>
</HTML>
```

3. Let's try addition of numbers entered via the prompt box:

```
<HTML>
<HEAD>
<TITLE>addition 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var number1, number2, answer;
    number1= prompt( "Please enter the first number");
    number2= prompt( "Please enter the second number");
    answer = number1 + number2;
    alert("the answer is: " + answer);
</SCRIPT>
</BODY>
</HTML>
```

The result here was not what was expected. Rather than adding the two numbers, it has appended or joined them together. This is because the + operator has a dual purpose: it is used to add numbers and append or join strings. Because numbers are entered in prompt boxes as strings rather than numbers, this can result in different values when arithmetic operations are performed on them. If you change the arithmetic to a multiplication, there will not be a problem because JavaScript is intelligent enough to recognise that the input is a number. It is nonsensical to multiply strings.

The solution? If you want the input to be treated as a number rather than a string, use the `parseInt ()` method. This method takes one parameter, which is the string you want to convert to an integer.

For example:

```
number = prompt("enter the number:");
intNumber = parseInt(number);
```

4. Try it out:

```
<HTML>
<HEAD>
<title>addition 2</title>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var number1, number2, answer;
    number1 = prompt("enter the first number:");
    intNumber1 = parseInt(number1);
    number2 = prompt("enter the second number:");
    intNumber2 = parseInt(number2);
    answer = intNumber1 + intNumber2;
    alert("the answer is: " + answer);
</SCRIPT>
</BODY>
</HTML>
```

*****Note:** This only works with the integer values you entered. Try entering decimal numbers like 12.7 and 13.4

To work with decimal numbers you need to use the function **parseFloat()**

Chaining

The two lines in the JavaScript above:

```
number1 = prompt("enter the first number:");
intNumber1 = parseInt(number1);
```

can be replaced by one as follows:

```
intNumber1 = parseInt(prompt("enter the first number:"));
```

This is called **Chaining Functions**. You are putting 2 or more functions in a row to chain them together. Use the JavaScript above to try it and see.

Making Statements

Comparison Operators:

Operator	Comparative Test
==	Equality
!=	Inequality
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Conditional IF

This is used to perform the basic conditional JavaScript test to evaluate an expression for a Boolean value. The statement(s) following the evaluation will only be executed when the expression returns true.

Example:

```
var msg, num;
num = 9;
if (num > 5) {
    msg = "This number is greater than 5";
    alert(msg);
}
```

The curly brackets { } take the place of the words **Then** and **End If** that we use in pseudo code and are used to define blocks of code

Takes the place of **End If**

Exercise 11 – Conditional Statements

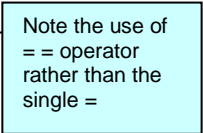
1. Try it out:

```
<HTML>
<HEAD>
<TITLE>comparison 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var msg, num;
    num = 9;
    if (num > 5) {
        msg = "This number is greater than 5";
        alert(msg);
    }
</SCRIPT>
<P> All done.
</BODY>
</HTML>
```

When you use the { or } brackets you do not place a semicolon (;) at the end of the line.

2. Try it out:

```
<HTML>
<HEAD>
<TITLE>password 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var password;
    password =prompt("Enter your password:");
    if (password == "secret") {
        alert("you have entered the correct password")
    }
</SCRIPT>
<P>All done.
</BODY>
</HTML>
```



Note the use of
== operator
rather than the
single =

3. Try the above with the password (secret) typed in uppercase (SECRET) as well as lowercase. Because **UPPERCASE** characters are treated as different to lowercase, the uppercase password will not work. Sometimes you may wish for the password to be accepted if the letters are correct, even if the case isn't. To do this, you can use the **string.toUpperCase()** or the **string.toLowerCase()** method.

Replace the comparison line:

```
if (password == "secret") {
```

with:

```
if (password.toLowerCase() == "secret") {
```

Now if you enter in the password in capitals (SECRET), it will be accepted.

Using Comments

Comments are used to explain what you are doing or trying to do in your code. They are used for yourself as the programmer or for others to help them understand what you are doing. They become part of the interior documentation of your program and the computer ignores what is written in them.

There are two different methods to put a comment into your JavaScript. If you want to just add a quick note at the end of a line of code you can use two forward slashes `//`. If you wish to make a note that extends over more than one line you use the multi-line comment `/* ... */`

If – Else Statements

The JavaScript keyword “else” can be used with an “if” statement to provide alternate code to execute in the event that the test expression returns false. This is known as conditional branching.

Exercise 12 – Using Comments

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>password 2</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var password;
    password = prompt ("Please enter your password");

    /* In this code I'm using the toLowerCase() function to
       make everything the user types on the keyboard appear
       as lower case letters
    */

    if (password.toLowerCase() == "secret" ) {
        alert("You have entered the correct password!");
    } else {
        alert("You have entered the wrong password!");
    } //end if

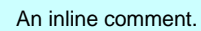
</SCRIPT>
<P> All done.
</BODY>
</HTML>
```

A multi-line comment to let me know what I'm doing in the code

An inline comment to let me know I've ended the **if - else**

2. Another example, try it out:

```
<HTML>
<HEAD>
<TITLE>comparison 2</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var number;          // Declaring a variable called number
    number = parseInt(prompt ("Please enter a number"));
    if (number > 5 ) {
        alert("the number is greater than 5!");
    } else {
        alert("the number is less than or equal to 5!");
    } //end if
</SCRIPT>
<P> All done.
</BODY>
</HTML>
```



An inline comment.

3. The **if-else** statement can be combined to provide more complicated results:

Try it out:

```
<HTML>
<HEAD>
<TITLE>comparison 3</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
    var mark;
    mark= parseInt(prompt("Enter your final mark:"));
    if (mark>= 85){                // If the mark is greater than or equal to
        alert("high distinction"); // 85 then write High Distinction
    }else if (mark >= 75) { //check if it is greater than or equal 75
        alert("distinction");
    } else if (mark >= 65){        // Multi line comments can also be
        alert("credit");          // written this way
    } else {
        alert("fail");
    } // end if
</SCRIPT>
</BODY>
</HTML>
```

Loops in Javascript

JavaScript provides a number of different types of loops. The two that we are using in this course are:

The Pre-Test Loop

The pre-test (**while**) loop is implemented in JavaScript as a **while**. The **while** loop, loops through a block of code as long as a specified condition is true.

```
set condition
while (condition) {
    ...
    ...(continue with statements)
    ...
    reset condition
}
```

Example 1

```
var num;
num = parseInt( prompt("enter a number greater than 1"));
while (num<=1) {
    alert("the number should be greater than 1");
    num = parseInt( prompt("enter a number greater than 1"));
}
```

Example 2

```
var num, sum;
var count = 0;
sum=0;
while (count < 4) {
    num = parseInt( prompt("enter a number greater than 1"));
    sum = sum + num;
    count = count +1;
}
alert("the sum of the numbers is " + sum);
```

The Post-Test Loop

The post-test (repeat) loop is implemented in JavaScript as a **do-while**.

```
set condition
do {
    ...
    ...(continue with statements)
    ...
} while (condition)
```

Example 3

```
var num;
do {
    num = parseInt( prompt("enter a number greater than 1"));
} while (num<=1)
...
...(continue with statements)
```

Example 4

```
var num, sum;
var count = 0;
sum=0;
do {
    num = parseInt( prompt("enter a number greater than 1"));
    sum = sum + num;
    count = count +1;
} while (count < 4)
alert("the sum of the numbers is " + sum);
```

More Than One Test Condition:

Syntax	Name	Operands	Results
&&	And	Boolean	Boolean
	Or	Boolean	Boolean
!	Not	One Boolean	Boolean

Example 5

To ensure that a number is greater than 0 and less than or equal to 100:

```
var num;
num = parseInt( prompt("enter a number greater than 1"));
while ((num <=0) || (num >100)) {
    alert( "number out of range");
    num = parseInt( prompt("enter a number greater than 1"));
}
```

Example 6

To ensure that a number is greater than 0 and less than or equal to 100:

```
var num;
do {
    num = parseInt( prompt("enter a number greater than 1"));
}while ((num <=0) || (num >100))
```

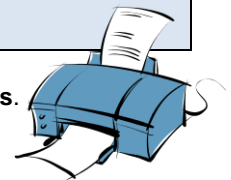
Exercise 13 – Using the WHILE Loop

```
<HTML>
<HEAD>
<TITLE>Comparison3 loop</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="TEXT/JAVASCRIPT">
var mark;
var count=0;
while (count<3){
    mark = parseInt(prompt("Please enter your final mark:"));
    if (mark >= 85){                //If the mark is greater than
        alert("High Distinction"); //or equal to 85 write HD
    }else if (mark >= 75){          //If Greater than or equal to
        alert("Distinction");      // 75 then write "Distinction"
    }else if (mark >= 65){          // same for 65 write "Credit"
        alert("Credit");
    }else {
        alert("Fail");             //If none of the above write "Fail"
    } //end if
    count=count+1;
}
</SCRIPT>
<p>
all done.
</BODY>
</HTML>
```

Exercise 14 – JavaScript Practice Exercises

Create scripts for each question you completed in pseudo code earlier in **Exercise 2 - Algorithms**.
Input will be via a prompt box.

Print the scripts for each question for your **PORTFOLIO**.



Defining and Calling Functions

A **function** is a piece of JavaScript code that can be executed once or many times by the JavaScript application.

A function definition has these basic parts:

- The function keyword.
- A function name.
- A comma-separated list of arguments to the function in parentheses.
- The statements in the function in curly braces.

It's important to understand the difference between **defining** a function and **calling** a function.

Defining a Function

Using the keyword “**function**” **defines** the function and tells the computer the name of the function and specifies what it does when the function is called.

Calling a Function

To use the function, we **call** the function by typing the function's name. **Calling** the function actually performs the specified actions with the indicated arguments.

Generally, you should define the functions for a page in the **HEAD** portion of a document. That way, all functions are defined before any content is displayed. Otherwise, the user might perform an action while the page is still loading that triggers an event handler and calls an undefined function, leading to an error.

The following example defines a simple function in the **HEAD** of a document and then calls it in the **BODY** of the document:

Exercise 15 - Functions

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>function 1</TITLE>
<SCRIPT TYPE="text/javascript">
    function call_alert() {
        alert("This is a simple function call");
    }
</SCRIPT>
</HEAD>

<BODY>
<SCRIPT>
    call_alert();
</SCRIPT><P> All done.

</BODY>
</HTML>
```

The JavaScript function is written between the <HEAD> and </HEAD> tags of the HTML document

And called between the <BODY> and </BODY> of the HTML document

In this example, a **function** has been declared, named `call_alert()`. The name must adhere to the naming conventions of variables. The name is always followed by a pair of plain brackets `()`, followed by a pair of curly brackets `{}`, which contains the code to be executed. In this case, the code to be executed calls the JavaScript `call_alert()` function, to display an alert dialog box containing a message.

The function can be called from anywhere in the document to execute the statement that it contains. Try calling it twice:

2. Try it out:

```
<HTML>
<HEAD>
<TITLE>function 2</TITLE>
<SCRIPT TYPE="text/javascript">

function call_alert() {
    alert("This is a simple function call");
}

</SCRIPT>
</HEAD>

<BODY>
<P> Let's call the function once:

<SCRIPT>
    call_alert();
</SCRIPT>

<P> Let's call it again!

<SCRIPT>
    call_alert();
</SCRIPT>

<P> All done.
</BODY>
</HTML>
```

The JavaScript function **call_alert()** is written between the <HEAD> and </HEAD> tags of the HTML document

And called twice in the body

Function Parameters and Arguments

The plain brackets that follow the name of all functions may be used to contain data for use in the code to be executed. (In the same way that the brackets in the `alert` function are used to display the text that you want to display.)

```
functionName(parameter1, parameter2, parameter3) {  
    code to be executed  
}
```

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** passed to (and received by) the function.

Exercise 16 – Function Parameters & Arguments

1. Try it out:

```
<HTML>  
<HEAD>  
<TITLE>function 3</TITLE>  
<SCRIPT TYPE="text/javascript">  
  
    function call_alert(msg) {  
        alert(msg);  
    }  
</SCRIPT>  
</HEAD>  
<BODY>  
<P> Let's call the function once:  
<SCRIPT>  
    call_alert("first call of the function");  
</SCRIPT>  
<P> Let's call it again!  
<SCRIPT>  
    call_alert("second call of the function");  
</SCRIPT>  
<P> All done.  
</BODY>  
</HTML>
```


2. Another example. Try it out:

```
<HTML>
<HEAD>
<TITLE>function 4</TITLE>
<SCRIPT TYPE="text/javascript">

function square(number) {
    return number * number; // this code returns the number squared
}

</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
    var number1, number2;
    number1 = square(5);
    number2 = square(6);
    alert("the square of 5 is: " + number1);
    alert("the square of 6 is: " + number2);
</SCRIPT>
<P> All done.
</BODY>
</HTML>
```

The function is called twice. Each time with a different argument and the answer is stored in a different variable and printed out in an alert box

The function square takes one parameter, called number. The function consists of one statement

```
return number * number
```

that indicates to return the parameter of the function multiplied by itself. The return statement specifies the value returned by the function. In the **BODY** of the document, the statement “**square(5)**” calls the function with an argument of five. The function executes its statements and returns the value twenty-five. The script displays two alert boxes:

“the square of 5 is 25”

“the square of 6 is 36 “

All done will appear in the web page.

3. Another example. Try it out:

```
<HTML>
<HEAD>
<TITLE>add function</TITLE>
<SCRIPT TYPE="text/javascript">

function add(a,b) {
    /* Two numbers are received and added together
    * then the result is sent back to part of the script
    * that called the function */

    return a+b;
} // end function add()

</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>

    var sum1, sum2, answer; // Declare all the variables we need
    sum1 = add(3,4); // Call the function add and store the result in sum1
    sum2 = add(4,5); // Call the function add and store the result in sum2

    // Call the function add and store the result in answer
    answer = add(sum1,sum2);

    alert(answer); // Show the answer in an alert box

</SCRIPT>
<P> All done.
</BODY>
</HTML>
```

Forms

Forms are used to collect different kinds of user input and are an area that can contain form fields or elements. These are objects that allow the visitor to enter information - for example:

- Text Boxes
- Textarea Fields
- Drop-Down Menus
- Checkboxes
- Radio Buttons
- Buttons

The user can click on to start or complete some action.

*****Note:** When the visitor clicks a submit button, the content of the form is usually sent to a program that runs on the server. However, since we won't be using a server we won't be using this button

FORM Tag

A form is defined with the **<FORM>** tag.

```
<FORM>
  <INPUT>
  <INPUT>
</FORM>
```

Main Settings: NAME

The **NAME** setting adds an internal name to the form so the program that handles the form can use it to identify any fields that are on it.

*****Note:** For our purposes we will only need the **NAME** setting of the **<FORM>** tag. Other settings are for use with a server which we will not be using in this subject

INPUT Tag

The most used form tag is the **<INPUT>** tag. The type of input is specified with the type attribute.

Text Fields - Text fields are used when you want the user to type letters, numbers, etc. in a form.

Main Settings: NAME, SIZE, MAXLENGTH, VALUE

The **NAME** setting adds an internal name to the field so the program that handles the form can identify the fields.



The **SIZE** option defines the width of the field. That is how many visible characters it can contain. The default setting is 20 characters.

The **MAXLENGTH** option defines the maximum length of the field. That is how many characters can be entered in the field. If you do not specify a **MAXLENGTH**, the visitor can easily enter more characters than are visible in the field at one time.

The **VALUE** setting defines what will appear in the box as the default value.

Exercise 17 – Input Tag

1. Try it out:

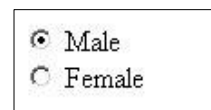
```
<HTML>
<HEAD>
<TITLE>Form 1</TITLE>
</HEAD>
<BODY>
<FORM>
  First name:
  <INPUT TYPE="TEXT" NAME="firstname">
  <BR>
  Last name:
  <INPUT TYPE="TEXT" NAME="lastname">
</FORM>
<P> All done.
</BODY>
</HTML>
```

*****Note** that the form itself is not visible. Also note that in most browsers, the width of the text field is 20 characters by default.

Radio Buttons

Radio Buttons are used when you want the user to select one of a limited number of choices.

Main Settings: **NAME**, **VALUE**



The **NAME** setting tells which group of radio buttons the field belongs to. When you select one button, all other buttons in the same group are unselected. If you couldn't define which group the current button belongs to, you could only have one group of radio buttons on each page.

The **VALUE** setting defines what will be submitted if checked.

Exercise 18 – Radio Buttons

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>Form 2</TITLE>
</HEAD>
<BODY>
<FORM>
  <INPUT TYPE="RADIO" NAME="sex" VALUE="male"> Male
  <BR>
  <INPUT TYPE="RADIO" NAME="sex" VALUE="female"> Female
</FORM>
<P> All done.
</BODY>
</HTML>
```

Checkboxes

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

Main Settings: **NAME**, **VALUE**, **CHECKED**

I have a bike: ☒
I have a car: ☐
I have an airplane: ☐

The **NAME** setting adds an internal name to the field so the program that handles the form can identify the fields.

The **VALUE** setting defines what will be submitted if checked.

CHECKED determines which Radio Button is already marked by Default.

Exercise 19 - Checkboxes

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>Form 3</TITLE>
</HEAD>
<BODY>
<FORM>
  I have a bike:
  <INPUT TYPE="CHECKBOX" NAME="vehicle" VALUE="Bike" CHECKED>
  <BR>
  I have a car:
  <INPUT TYPE="CHECKBOX" NAME="vehicle" VALUE="Car">
  <BR>
  I have an airplane:
  <INPUT TYPE="CHECKBOX" NAME="vehicle" VALUE="Airplane">
</FORM>
<P> All done.
</BODY>
</HTML>
```

Text Areas

Text areas are text fields that can span several lines. Unlike most other form fields, text areas are not defined with an `<INPUT>` tag. Instead you enter a `<TEXTAREA>` tag where you want the text area to start and a closing `</TEXTAREA>` tag where you want the area to end.



Main Settings: ROWS, COLS, NAME, WRAP

The **COLS** and **ROWS** settings are straightforward and simple. They specify how many columns and rows you want in your text area.

The **NAME** setting adds an internal name to the field so the program that handles the form can identify the fields.

WRAP has 3 options **OFF**, **VIRTUAL** and **PHYSICAL** and is the trickiest part of text areas.

- If you turn **WRAP OFF** the text is handled as one long sequence of text without line breaks.
- If you set it to **VIRTUAL** the text appears on your page as if it recognized line breaks - but when the form is submitted the line breaks are turned off.
- If you set it to physical the text is submitted exactly as it appears on the screen - line breaks included

Exercise 20 - Textareas

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>Form 4</TITLE>
</HEAD>
<BODY>
<FORM>
  This is outside the area<P>
  <TEXTAREA COLS="40" ROWS="5" NAME="myname">
    Now we are inside the area - which is nice.
  </TEXTAREA>
  <P>
    And now we are outside the area again.
</FORM>
<P> All done.
</BODY>
</HTML>
```

Everything written between the `<TEXTAREA ...>` `</TEXTAREA>` tags will be presented in the text area box.

document	refers to the webpage the form is in
form	refers to the user-defined name of the FORM (in this case, nameForm)
object	of the control placed on the form (in this case the TEXT box called nameBox).
property	is the value of the object (in this case the text that has been typed into the TEXT box).

Exercise 22 – More JavaScript Problems

1. Use the form, text box and button to enter in a number and display that number.
2. Change this to enter in a number and display that number as a percentage.
3. Change this to enter in a number and display:
 “A” if the number is greater than or equal to 85
 “B” if the number is greater than or equal to 75
 “C” if the number is greater than or equal to 65
 “D” if the number is greater than or equal to 50
 otherwise display “fail”
4. Create a form that displays the names and prices of two products – a toaster priced at \$54.56 and a sandwich maker priced at \$98.99 - and two text boxes that request the user to enter in the quantity of each product that he or she wishes to order. You are then to display the amount owing. Assume that the user wishes to order both products.

Exercise 23 – JavaScript Practice Exercises Using Objects

Try and change each of the JavaScript Practice Exercises completed in **Exercise 13** so that they accept their input via text boxes and display their answer when the user presses a button.

“For” Loops

The “**for**” loop is used when you know in advance how many times the script should run. There are **four** important aspects of a JavaScript **for** loop:

1. The counter variable is something that is created and usually used only in the **for** loop to count how many times the for loop has run.
2. The conditional statement that decides whether the **for** loop continues executing or not. This check usually includes the counter variable in some way.
3. The counter variable is incremented after every loop in the increment section of the **for** loop.
4. The code that is executed for each loop through the **for** loop.

This may seem strange, but steps 1-3 all occur on the same line of code. This is because the for loop is such a standardized programming practice that the designers felt they might as well save some space and clutter when creating the for loop.

JavaScript for Loop Example

This example will show you how to create a simple for loop that prints out the value of our counter until the counter reaches 5. Pay close attention to the three different items that are on the first line of the for loop code. These are the important for loop parts 1-3 that we talked about.

Exercise 24 – “For” Loop

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>For Loop 1</TITLE>
</HEAD>
<BODY>
<SCRIPT TYPE="text/javascript">
<!--                                     // this is an HTML comment tag
var linebreak = "<br />";              // Set up the HTML code for a new line
document.write("For loop code is beginning");
document.write(linebreak);

for(i = 0; i < 5; i++) {
    document.write("Counter i = " + i);
    document.write(linebreak);
} // end for loop

document.write("For loop code is finished!");
close the HTML comment    -->
</SCRIPT>
<P> That's all
</BODY>
</HTML>
```

The counter is set [i = 0]
The counter is checked [i < 5]
The counter is incremented [i++]

Getting Values Out Of Radio Buttons

To get the value from a radio button you must check all the buttons in a group to see which one was checked. This requires the use of a **for** loop to step through each button in the group in order.

Exercise 25 – Getting Values Out of a Radio Button

1. Try it out:

```
<HTML>
<HEAD>
<TITLE>Radio Button Exercise</TITLE>

<script type="text/javascript">
var radioValue;

function getRadioValue(radioObj){
    var numButtons = radioObj.length; // Get the number of buttons in the group
    for(var i = 0; i < numButtons; i++) {
        if(radioObj[i].checked) { // see if this button is the one that was
clicked
            radioValue = (radioObj[i].value); // if so capture the value of the button
        } //end if
    } // end for
} // end function getRadioValue()

function showChoice() {
alert("You chose " + radioValue);
}
</script>
</HEAD>
<BODY>
<FORM NAME='myForm'>
    Choose a Radio Button and click "Show Button Number"<p>
    <input type="radio" name="someRadio" value="Button 1"
onchange="getRadioValue(document.myForm.someRadio)"> Button 1<br>
    <input type="radio" name="someRadio" value="Button 2"
onchange="getRadioValue(document.myForm.someRadio)"> Button 2<br>
    <input type="radio" name="someRadio" value="Button 3"
onchange="getRadioValue(document.myForm.someRadio)"> Button 3<br>
    <input type="radio" name="someRadio" value="Button 4"
onchange="getRadioValue(document.myForm.someRadio)"> Button 4<br>
    <input type="radio" name="someRadio" value="Button 5"
onchange="getRadioValue(document.myForm.someRadio)"> Button 5<br>
    <p>
    <input type="button" value="Show Button Number" onclick="showChoice()">
</FORM>
</BODY>
</HTML>
```

Elements of JavaScript

JavaScript code is made up of statements, which serve to make assignments, compare values, and execute other sections of code. Below is a chart summarizing the main elements of JavaScript grammar.

Variables	A variable is used to store data for manipulation within a JavaScript program. A variable has a name and a value. Example: total may possess a value of 100. total=100;
Operators	Operators can be used to calculate or compare values. Example: Two values may be summed using the addition operator (+); total+tax Example: Two values may be compared using the greater-than operator (>); total>200
Expressions	Any combination of variables, operators, and statements that evaluate to some result. Example: total=100; (worded as total has the value 100) Example: if (total>100) (worded as if the value of total is greater than 100.)
Statements	JavaScript statements may take the form of conditionals, loops, or object manipulations. Statements need to be separated by semicolons if multiple statements reside on the same line. Example: if (total>100) {statements;} else {statements;} Example: while (clicks<10) {statements;}
Objects	An object may contain many properties, each property acts like a variable reflecting a certain value. JavaScript can reference a large number of "built-in" objects that refer to characteristics of a Web document. For instance, the document object contains properties that reflect the background colour of the current document, its title, and many more.
Functions and Methods	A function is a discrete set of statements that perform some action. It may accept incoming values (parameters), and it may return an outgoing value. A function is "called" from a JavaScript statement to perform its duty. For example, alert("hello") is a function that displays an alert box on the screen, and displays the text "hello" in the alert box. A method is simply a function that is contained in an object. For instance, a function which closes the current window, named close(), is part of the window object; thus, window.close() is known as a method in the object window.