PROGRAMMING WITH LIVECODE Community Edition

What can I make with LiveCode?

LiveCode is used to create apps, games, interactive ebooks and comics.

LiveCode is used to create powerful in-house systems and mobile apps for public and private sector organizations.

Higher Computing Science

MATERIALS PRODUCED AT GHS BY MR S. WHYTE

The Software Development Process

Introduction

The Software Development Process (**SDP**) can be split into **7 main** steps which are carried out in **order**. These steps should be carried out when creating **any** programming project and are summarised below.

Analysis — A

A statement about **what** your program is going to do. The analysis stage will also cover areas of **feasibility**, i.e. is there enough **time** and **money** available to complete the project?

Design — Dance

This involves designing both the **user interface** and the **structure** of the **program code**.

For the purpose of Higher Computing, more emphasis will be placed on designing the **structure** of the **program code** rather than the design of the user interface. We will be using a design notation known as **pseudocode** to achieve this. More is mentioned about pseudocode on the next page.

Implementation — In

The implementation stage involves keying in the program code using the built in **text editor** within the programming environment. We will use LiveCode to create our programs.

Testing — The

Testing is an important part of any project. Testing ensures that your program is **reliable** and **robust** in the sense that it should produce the **correct results** and **not crash** due to **unexpected input**.

We should test our program with **three** sets of test data. These are:

- **Normal** (accepted data within a set range)
- **Extreme** (accepted data on the boundaries)
- **Exceptional** (data that is not accepted).

Documentation — Dark

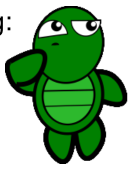
Documentation is usually produced in the form of a **user guide** and a **technical guide**. The user guide shows the user how to **use** the **functions** and **features** of the **software** whereas the **technical guide** gives the user information on how to **install** the **software** as well as the **minimum system requirements**.

Evaluation — Every

An evaluation is usually a **review** which shows that your program is **fit for purpose**, in other words, it does **exactly** what it was **designed** to do.

The evaluation should also focus on the **readability** of your program code. For example, if **another programmer** was asked to **maintain** your program code at a later date, would they be able to understand what was going on? You should always ensure your program is **readable** by doing the following:

- **Use** of **meaningful identifiers** for **variable** and **array names**
- **Use** of **internal commentary** *(// This subroutine will do the following....)*
- **Effective** use of **white space** between **subroutines** to **space out** the **program**.
- **Indentation** to show the **start** and **end** of any control structures such as a fixed **loop**.
- **Parameter passing** to show the programmer what **variables** and **arrays** are being **passed in** and **out** of each **subroutine** and which **parameters** are being **changed**.



Maintenance — Monday

Maintenance is performed at the **very end** of the project. You will not be required to perform any maintenance on your programs but you will need to know about **Corrective**, **Adaptive** and **Perfective** maintenance. These are covered in the Software Development theory notes.

The Design Process

Pseudocode and Data Flow

The design of a program is **very important** as it allows the programmer to **think** about the **structure** of the program **before** they begin to **create** it.

The most common way to design the logic of a program is to use a text-based notation known as **Pseudocode**. **Pseudocode** is a cross between **programming language** and our own **English language**. It makes a program **easier** to **understand** without relying on the use of a programs complex **commands** and **syntax**.

The design is built up of two parts, the **first** is the **Stepwise design**. This shows the **main steps** of the program. The **second** part is the **Stepwise Refinement**. This involves **breaking** these **main steps** into even **smaller steps** so eventually, **one line** of **pseudocode** becomes **one line** of **program code**.

Here is the program pseudocode to calculate the volume of a room using the variables **length**, **breadth**, **height** and **room_volume**. Study both the **pseudocode** and **data flow** very closely to understand what is going on:

Stepwise Design *(the **main steps** of the program)*

1.	**Setup the global variables**	No Data flow required
2.	**Initialise variables**	No Data flow required
3.	**Get room measurements**	**In/Out**: room_length, room_breadth, room_height

Data flow explanation for step 3: Since room_length, room_breadth and room_height will have been initialised to 0 in the subroutine initialise, they are coming into this step with the value 0 and will be given new values according to the size of the room. Hence they are **IN** as 0's and then passed as **OUT** as a new value.

4.	**Calculate Room Volume**	**In**: room_length, room_breadth, room_height **In/Out**: room_volume

Data flow explanation for step 4: The room_length, room_breadth and room_height variables are passed **IN** to be used in the calculation for room_volume. As a result, they are not changing their values so are just passed as **IN's**. The room_volume variable would have been set to 0 in the initialise subroutine and as a result of the calculation, room_volume will be given a **new** value so it's **IN/OUT**.

5.	**Display Room Volume**	**In:** room_volume

Data flow explanation for step 5: Only the room_volume is to be displayed and it is **not** changing in value from what was calculated in step 4, so it is just an **IN** variable within this subroutine.

Stepwise Refinement *(the **main steps further refined** into smaller steps)*

```
1.      Setup variables
1.1     Setup room_length, room_breadth, room_height and room_volume as global variables

2.      Initialise variables
2.1     Put 0 into room_length, room_breadth, room_height and room_volume

3.      Get room measurements
3.1     Ask the user for the length of the room in metres
3.2     Put it into the variable room_length
3.3     Ask the user for the breadth of the room in metres
3.4     Put it into the variable room_breadth
3.5     Ask the user for the height of the room in metres
3.6     Put it into the variable room_height

4.      Calculate room volume
4.1     Put room_length * room_breadth * room_height into room_volume

5.      Display room volume
5.1     Put a message telling the user the volume of the room in cubic metres using the variable room_volume
```

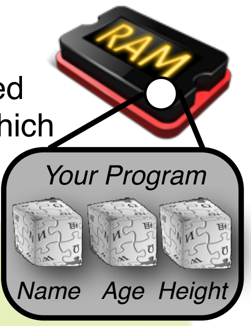
Stepwise Refinement:
The main steps are broken down further (refined). We use 3.1, 3.2, 3.3, etc.

Notice that the pseudocode looks **more** like our own language rather than that of the programs.

What are Variables?

Implementation

Let's talk about variables as they are **very** important in programming.



*Variables are **identifiers** in **RAM** **used** to store **data** in a running program.*

To put it simply, a variable is like a "box" into which data can be placed whilst a program is **running**. We give them names (identifiers) which suggest or give us a clue as to what data is being held in the variable.

Variables can be store **different types** of **data**, for example:

- **Text** (known as **strings**), e.g. *Steven*, *Jim*, or *Lisa* etc.
- **Real numbers**, (numbers with a **decimal point**) e.g. *3.14*, *5.7* or *11.16*, etc.
- **Integer numbers**, (**whole** numbers) e.g. *5*, *7* or *102*, etc.
- **Two state values** (known as **boolean**), e.g. Yes/No, True/False, 1/0, etc.

Here are the **variables** you will use in your first program:

```
global room_length, room_breadth, room_height, room_volume
```

Some languages allow variables to be Setup at **any point** in a program. Other languages like LiveCode require **global variables** to be declared at the **start** of the program before they can be used.

The **advantages** of **declaring variables** at the **start** are as follows:

- It allows the translator program to **reserve suitable areas of memory** to hold the data structures which will subsequently be used by the program.
- The declaration of variables serves as good **discipline** to programmers because they have to create a **list** which details the **name** and **purpose** of **each variable** used in their program.

Variable Rules

Variables **cannot** contain any **spaces** and must **not** be a **reserved command** in LiveCode. You can tell if a variable has been accepted as it will appear in **black font** when it is typed into the text editor as shown below:

```
ask "Please enter the length of the room in metres"

put it into room_length
```

$\boxed{\text{room\_length}} \longleftarrow$ *This is the variable*

The ampersand **separates** both the **variable** and the **text** to be printed on the screen. Two ampersands **&&** together will also include a single space when the text is printed. For example the following code:

```
put "The volume of this room is" &&room_volume&& "cubic metres." into field "output"
```

....will produce:

"The volume of this room is 3000 cubic metres."

Classification of Variables

Variables fall into **two** main types. The type of a variable determines **where** it can be **used** in a **program**.

The **two** main types of variable are **local variables** and **global variables**. A description of each is given below. It is important that you understand the difference as you will gain experience of using **both types** of **variable** when you are programming.

Local Variables

A **local variable** is one which only exists within **one subroutine**, **function** or **procedure**.

Local variables are **created** when the subroutine is called (run) and are then **destroyed** when the subroutine **terminates**. They **cannot** be accessed or assigned a value except within that **subroutine**.

The example below shows the use of a local variable:

```
on get_users_name
  //  Setup the local variable to be used in this subroutine
  local key_pressed

  repeat until key_pressed = "Y" or key_pressed = "y"
    ask "Please enter your name"
    put it into the_name_of_person
    ask "Are you happy with the name entered? (Y or y for yes)"
    put it into key_pressed
  end repeat
end get_users_name
```

In the subroutine **get_users_name**, the local variable **key_pressed** is created. The purpose of this variable is to check whether or not the user is happy with the name that they have entered by keying in “Y” or “y”, otherwise the program will keep looping. This local variable is unique to this subroutine and **cannot** be used in any other subroutine.

The **advantage** of using **local variables** is that it **prevents** them from being used **elsewhere** in the program and possibly having their **contents accidentally changed**.

Global Variables

A **global variable** is one which can be **accessed** and **altered** from **any** part of the program, even from another script/event so long as it is **declared** at the very **start**.

Global variables should **always** be used with **care** as their values may accidentally change if the programmer forgets that they have already used them in another subroutine. The example below shows the setting up of a series of global variables in LiveCode:

```
//  Setup the global variables to be used in this event
global name_of_person, age_of_person, address_of_person
```

In the code snippet above **three global variables** have been created. These variables can be used in **any subroutine** and in **any** LiveCode **event** so long as they are **declared** at the **start** of the **event** in the same way as shown above.

Parameter Passing

What is a Parameter?

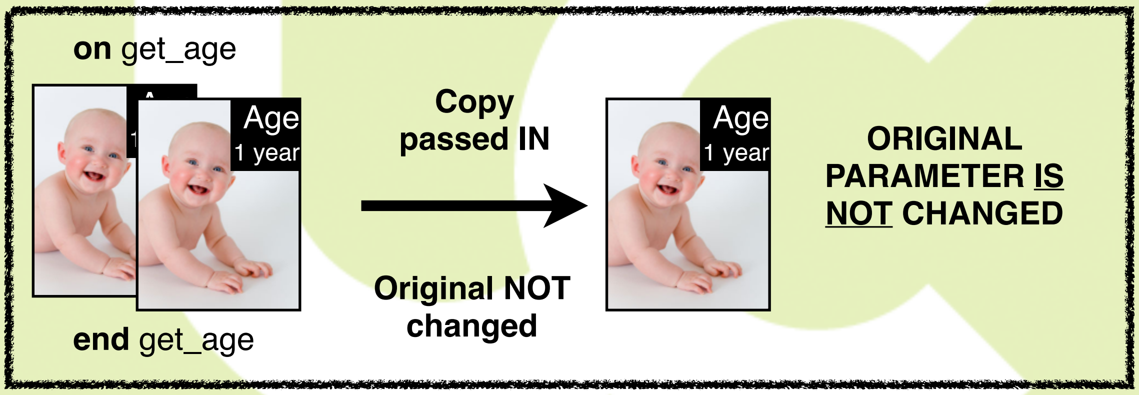
A **parameter** can either be a **variable** or an **array**. When a parameter is used, it can be passed into a sub-routine and **not changed** (passes by value) or passed into a subroutine and **changed** (passed by reference). Only **global variables** and **arrays** can be parameter passed because (as you have already learned), **only** a parameter that is **global** can be used in **more** than **one subroutine**.

For Higher Computing, you need to demonstrate both parameter passing by **value** and by **reference** within the programs you create. It is **vital** you **understand** how it works. Parameter passing works in the **same** way as the **data flow** you do during the **design**.

Parameter Passing by Value

Passing a parameter by **value** is used when a parameter is needed in a subroutine but its value **is not** going to **change** in the **subroutine**.

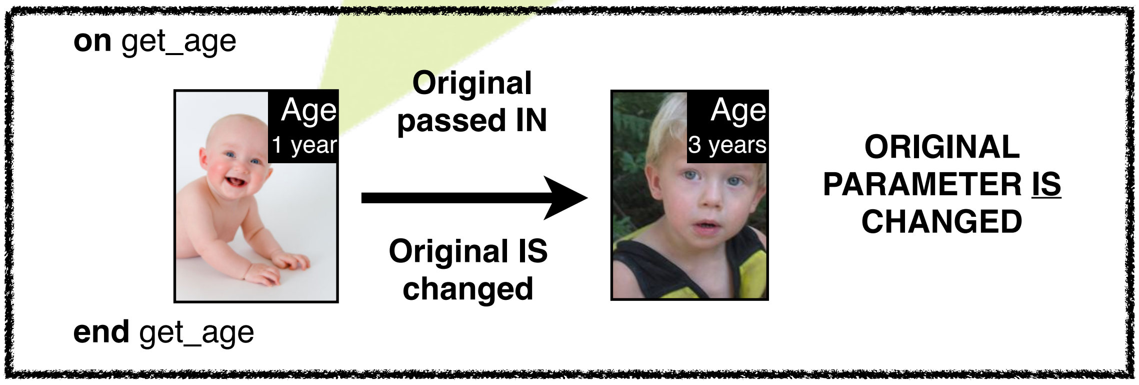
The subroutine will be passed a **copy** of the original parameter, so that the original parameter remains unchanged.



Parameter Passing by Reference

Passing a parameter by **reference** is used when a parameter is needed in a subroutine and its value **is** going to **change** in the **subroutine** when it is passed in.

The subroutine will be passed the original parameter and any changes made in the subroutine **will** result in a **change** to the original value(s) held within the parameter.



Parameter Passing (an example)

For Higher Computing, you need to demonstrate **both** of these methods of **parameter passing** within **all** programs you create. Study the program below carefully. This program calculates the volume of a room and it will be the first program you create. It **includes** parameter passing indicated in highlighted sections:

```
//  Here are our global parameters (variables) to be used in this event.
global room_length, room_breadth, room_height, room_volume

on mouseUp
//  After the names of each subroutine, we include all the global parameters used
//  within that subroutine separated by a comma.
   initialise
   get_room_measurements room_length, room_breadth, room_height
   calculate_room_volume room_length, room_breadth, room_height, room_volume
   display_room_volume room_volume
end mouseUp

on initialise
//  The first subroutine does not normally include any parameter passing as this
//  involves setting up the parameters to null or 0.
   put 0 into room_length
   put 0 into room_breadth
   put 0 into room_height
   put 0 into room_volume
end initialise

//  After the subroutine name below, you will notice that the parameter names have
//  an @ symbol before their name.  This indicates that the parameters are being
//  passed into this subroutine by reference, in other words, they are changing from 0
//  (initialised state) to whatever the user enters.
on get_room_measurements @room_length, @room_breadth, @room_height
   ask "Please enter the length of the room in metres"
   put it into room_length
   ask "Please enter the breadth of the room in metres"
   put it into room_breadth
   ask "Please enter the height of the room in metres"
   put it into room_height
end get_room_measurements

//  After the subroutine name below you will notice that the most of the parameters
//  are now being passed by value (no @ sign before the name).  This is because the
//  values have already been assigned in the previous subroutine and we do not want
//  them to change when passed into this subroutine.
//
//  The only value which is passed by reference is room_volume as it will be changed
//  from its initialised state of 0 to the result of the calculation below.
on calculate_room_volume room_length, room_breadth, room_height, @room_volume
   put (room_length * room_breadth * room_height) into room_volume
end calculate_room_volume

//  The only parameter which passed into this subroutine is room_volume.  This is
//  passed by value as it is the result of the calculation in the previous subroutine and
//  we do not want the parameters value to change.
on display_room_volume room_volume
   put "The volume of this room is" &&room_volume&& "cubic metres." into field “output”
end display_room_volume
```

LiveCode

LiveCode is a modern programming environment that has been created by an Edinburgh-based company called Runtime Revolution, **www.runrev.com**.

LiveCode is advertised as being a **very high level language** and is considered to be **even closer** to the way we speak and write as opposed to the sometimes **complex commands** and **syntax** used in other high-level programming environments.

Users can use LiveCode to create **any** type of program. This could range from a simple application which performs addition to a more advanced game application that could be run on a desktop computer or mobile phone.

LiveCode is an **event-driven programming language** which means that it involves the triggering of **events** such as a mouse click on a button or text entry into an output field.

The LiveCode programming environment can run on a **variety** of operating system **platforms**. This includes a **PC** running Windows XP, Vista, Windows 7 or Linux as well as on a **Mac** running OS X.

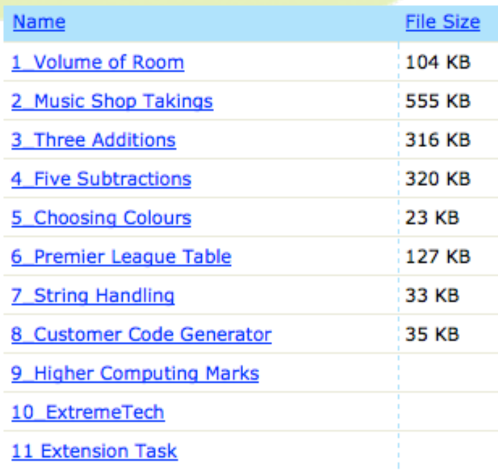
At least **400MB** of **hard disk** space and **256MB** of **RAM** is required in order for the programming language to run.

The LiveCode programming environment has already been **installed** in the **Applications** folder:

IMPORTANT

You will need to access to **Glow** in order to allow you to download the template stacks for each of the **LiveCode Programming Tasks**. You teacher will show you how to do this.

You should save each program you complete into your own folder as you may find that you need part of a program again to help you with your final assessment.

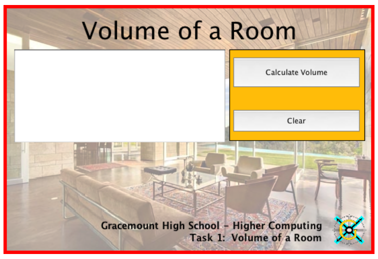


Name	File Size
1 Volume of Room	104 KB
2 Music Shop Takings	555 KB
3 Three Additions	316 KB
4 Five Subtractions	320 KB
5 Choosing Colours	23 KB
6 Premier League Table	127 KB
7 String Handling	33 KB
8 Customer Code Generator	35 KB
9 Higher Computing Marks	
10 ExtremeTech	
11 Extension Task	

Task 1: Volume of a Room

Specification

A program is required to calculate the volume of a room. The user will be asked for the length, breadth and height of the room in metres and then once calculated, the program will display the volume of the room in cubic metres.



Design: Pseudocode for "Calculate Room Volume" Button

Stepwise Design *(the main steps of the program with data flow)*

1.	Setup the global variables	
2.	Initialise variables	
3.	Get room measurements	**In/Out**: room_length, room, breadth, room_height
4.	Calculate room volume	**In**: room_length, room_breadth, room_height **In/Out**: room_volume
5.	Display room volume	**In**: room_volume

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup the global variables
1.1   Setup room_length, room_breadth, room_height and room_volume as global
      variables

2.    Initialise variables
2.1   Put 0 into room_length, room_breadth, room_height and room_volume

3.    Get room measurements
3.1   Ask the user for the length of the room in metres
3.2   Put it into the variable room_length
3.3   Ask the user for the breadth of the room in metres
3.4   Put it into the variable room_breadth
3.5   Ask the user for the height of the room in metres
3.6   Put it into the variable room_height

4.    Calculate room volume
4.1   Put the room_length * room_breadth * room_height into the variable room_volume

5.    Display room volume
5.1   Put a message telling the user the volume in cubic metres using the variable
      room_volume in the field "output"
```

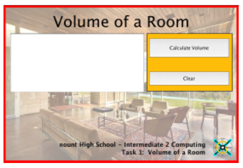
Implementation

After reading through the above design carefully, you are now ready to begin producing your program code.

Key the code in over the page **carefully** and **correct** any coding errors that you make.

Task 1: Volume of a Room

Implementation (continued)



Open the "**Volume of a Room**" stack. It can be found on Glow:

LiveCode Programming Tasks > 1_Volume of a Room.livecode

Enter the script below carefully into the **Calculate Volume** button. Check that the program works correctly by keying in some test data. See if the **same result** is produced if you key in the same numbers using a **calculator**.

```
// Setup the global variables to be used in this event
global room_length, room_breadth, room_height, room_volume

on mouseUp
  initialise
  get_room_measurements room_length, room_breadth, room_height
  calculate_room_volume room_length, room_breadth, room_height, room_volume
  display_room_volume room_volume
end mouseUp

on initialise
  // Initialise the global variables to 0
  put 0 into room_length
  put 0 into room_breadth
  put 0 into room_height
  put 0 into room_volume
end initialise

on get_room_measurements @room_length, @room_breadth, @room_height
  // Get the room measurements from the user
  ask "Please enter the length of the room in metres"
  put it into room_length
  ask "Please enter the breadth of the room in metres"
  put it into room_breadth
  ask "Please enter the height of the room in metres"
  put it into room_height
end get_room_measurements

on calculate_room_volume room_length, room_breadth, room_height, @room_volume
  // Calculate the volume of the room
  put (room_length * room_breadth * room_height) into room_volume
end calculate_room_volume

on display_room_volume room_volume
  // Display the volume of the room using the result within the room_volume variable
  put "The total volume of this room is" &&room_volume&& "cubic metres." into field "output"
end display_room_volume
```

So, what have we learned so far?

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

The LiveCode program area has three areas:

1. The **variable** list - lists all variables used in the program
2. The **event** list - this is a list of **all subroutines** which are run when the event is triggered by the user.
3. The **subroutines** - contain the **lines** of **code** to be **executed**.

ASK is a command that allows the programmer to ask the user a question or ask the user for a response. For example:

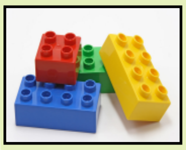
```
ask "Please enter the length of the room in metres"
```

PUT is a command that allows the programmer to transfer the users response **(it)** into a meaningful **variable**. For example:

```
put it into room_length
```

// are used to put **internal commentary** into a program or to space out different parts of the program to make it easier to read. For example:

```
// Display the volume of the room room
```



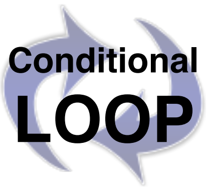
on and **end** are used to **start** and **end** of a subroutine. A subroutine must be started and ended, for example:

```
on display_room_volume room_volume
  put "The room volume is" &&room_volume into field “output”
end display_room_volume
```

One way to get one or more lines of code to repeat is by using a **loop**. The **two** main types of loop are a **fixed** loop and a **conditional** loop.

A **REPEAT WITH loop** can be used to repeat a piece of code as many times as the user sets it up for. In the example below, the loop is **fixed** at repeating the message “Hello World!” **4 times only**.

```
repeat with loop = 1 to 4
   put  "Hello Word!"
end repeat
```



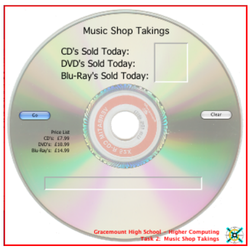
A **REPEAT UNTIL loop** can be used to repeat a line of code until a certain **condition** is met. In the example below, the loop will **not** finish **until** the user enters a **valid number** between **0 and 100**. This is the **condition**.

```
ask "Please enter a number between 0 and 100.”
put it into number
repeat until number >= 0 and number <= 100
  ask "Invalid number.  Please re-enter a number between 0 and 100."
  put it into number
end repeat
```

Task 2: Music Shop Takings

Specification

A program is required take in the number of CD’s, DVD’s and Blu-Ray Disks sold over the course of a day. The program will then find the combined total takings of CD’s (£7.99), DVD’s (£10.99) and Blu-Ray Disks (£14.99) sold and display this in an output field.



Design: Pseudocode for “Go” Button

Stepwise Design *(the main steps of the program)*

1.	Setup the global variables		
2.	Initialise variables		
3.	Get Number of Items Sold	**In/Out**: cds_sold, dvds_sold, blurays_sold	
4.	Calculate Total Takings	**In:** cds_sold, dvds_sold, blurays_sold	**In/Out**: total_takings
5.	Display Total Takings	**Out**: cds_sold, dvds_sold, blurays_sold, total_takings	

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup the global variables
1.1     Setup cds_sold, dvds_sold, blurays_sold and total_takings as global variables

2.    Initialise variables
2.1     Put 0 into cds_sold, dvds_sold, blurays_sold and total_takings

3.    Get Number of Items Sold
3.1     Ask the user for the number of CD’s sold
3.2     Put it into the variable cds_sold
3.3     Ask the user for the number of DVD’s sold
3.4     Put it into the variable dvds_sold
3.5     Ask the user for the number of Blu-Ray’s sold
3.6     Put it into the variable blurays_sold

4.    Calculate Total Takings
4.1     Put cds_sold * 7.99 +  dvds_sold * 10.99 + blurays_sold * 14.99 into the variable total_takings

5.    Display Total Takings
5.1     Set the number format to pounds and pence (00.00)
5.2     Put a message showing user the number of CD’s sold using the variable cds_sold in line 1
        of the field “output1”
5.3     Put a message showing user the number of DVD’s sold using the variable dvds_sold in line
        2 of the field “output1”
5.4     Put a message showing user the number of Blu-Ray’s sold using the variable blurays_sold
        in line 3 of the field “output1”
5.5     Put a message to the user showing them the total profit for that day using the variable
        total_takings in field “output2”
```

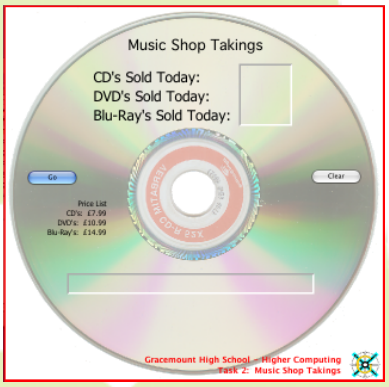
Task 2: Music Shop Takings

Implementation

Open the “**Music Shop Takings**” stack. It can be found on Glow:

LiveCode Programming Tasks > 2_Music Shop Takings.livecode

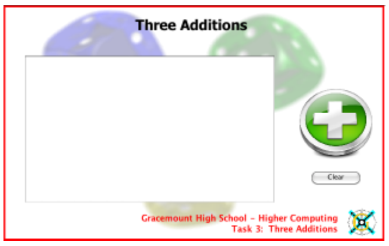
Copy the script below carefully into the “**Go**” button. Check that the program works correctly by keying in some test data. See if the same result is produced if you key in the **same** numbers using a **calculator**.



```
//  Setup the global variables to be used in this event
global cds_sold, dvds_sold, blurays_sold, total_takings

on mouseUp
   initialise
   get_number_of_items_sold cds_sold, dvds_sold, blurays_sold
   calculate_total_takings cds_sold, dvds_sold, blurays_sold, total_takings
   display_total_takings cds_sold, dvds_sold, blurays_sold, total_takings
end mouseUp

on initialise
   // Initialise the variables
   put 0 into cds_sold
   put 0 into dvds_sold
   put 0 into blurays_sold
   put 0 into total_takings
end initialise

on get_number_of_items_sold @cds_sold, @dvds_sold, @blurays_sold
   //  Get the number of CDs, DVDs and Blu-Ray Disks sold
   ask "Please enter the number of CD's sold today: "
   put it into cds_sold
   ask "Please enter the number of DVD's sold today: "
   put it into dvds_sold
   ask "Please enter the number of Blu-Ray's sold today: "
   put it into blurays_sold
end get_number_of_items_sold

on calculate_total_takings cds_sold, dvds_sold, blurays_sold, @total_takings
   //  Calculate the total takings by taking the amount of items sold by the user
   put cds_sold * 7.99 + dvds_sold * 10.99 + blurays_sold * 14.99 into total_takings
end calculate_total_takings

on display_total_takings cds_sold, dvds_sold, blurays_sold, total_takings
   //  Display the quantity of each type of produce sold during the day
   put cds_sold into line 1 of field "output1"
   put dvds_sold into line 2 of field "output1"
   put blurays_sold into line 3 of field "output1"

   //  This function sets the output of the total takings to two decimal places
   set numberformat to "00.00"

   //  Display the total cost
   put "Today you have made a total of £" & total_takings into field "output2"
end display_total_takings
```

Task 3: Three Additions

Specification



A program is required to test basic addition. The program will ask the user for their name, check to see if it's acceptable, then ask the user to answer **three** simple **additions** of two numbers (between **0** and **9**). The answer will then be checked by the program and a comment about the answer will be displayed.

The number correct out of three along with a comment should be displayed in the output field.

Design: Pseudocode for "Plus" graphic

Stepwise Design *(the main steps of the program with data flow)*

```
1.  Setup the Global Variables
2.  Initialise variables
3.  Get the Users Name      In/Out: name_of_person
4.  Three Additions         In/Out: first_number, second_number, my_answer  Out: name_of_person
```

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup the Global Variables
1.1     Setup name_of_person, first_number, second_number, my_answer as global variables

2.    Initialise variables
2.1     Put 0 into first_number, second_number and my_answer
2.2     Put "" into name_of_person

3.    Get the Users Name
3.1     Setup key_pressed as a local variable
3.2     Start a Repeat loop until the key_pressed = "Y" or "y"
3.3       Ask the user for their name
3.4       Put it into name_of_person
3.5       Ask the user if they are happy with the name entered
3.6       Put it into key_pressed
3.7     End Repeat

4.    Three Additions
4.1     Setup number_correct as a local variable
4.2     Put 0 into number_correct
4.3     Start a Repeat with question_number=1 to 3
4.4       Put a random number between 1 and 9 into first_number and second_number
4.5       Ask for the answer to first_number + second_number
4.6       If the cancel button is pressed then exit to the top of the program
4.7       Put it into my_answer
4.8       Put my_answer into line question_number of the field "output"
4.9       If my_answer =first_number + second_number then
4.10        Put a pop up message to the user saying, correct, well done!
4.11        Add 1 to number_correct
4.12      Else
4.13        Put a pop up message to the user saying, wrong answer.
4.14      End If
4.15    End Repeat
4.16    Put how many questions out of 3 the user got correct into line 5 of field "output"
4.17    If the number_correct is = 0 then put "very disappointing" into line 7 of field "output"
4.18    If the number_correct is = 1 then put "disappointing" into line 7 of field "output"
4.19    If the number_correct is = 2 then put "good work" into line 7 of field "output"
4.20    If the number_correct is = 3 then put "well done" into line 7 of field "output"
```

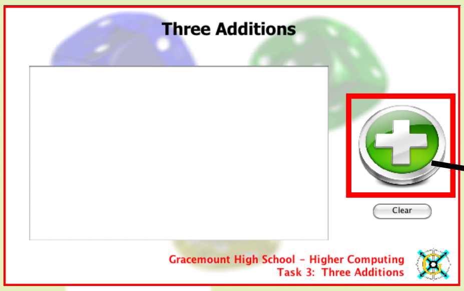
Task 3: Three Additions

Implementation

Open the "**Three Additions**" stack. It can be found on Glow:

LiveCode Programming Tasks > 3_Three Additions.livecode

Copy the script below carefully into the "**Plus**" graphic. Check that the program works correctly by keying in some test data. See if the same result is produced if you key in the same numbers using a calculator.



Plus graphic
(a button can also be a graphic)

```
// Setup the global variables to be used in this event
global name_of_person, first_number, second_number, my_answer

on mouseUp
  initialise
  get_users_name name_of_person
  three_additions name_of_person, first_number, second_number, my_answer
end mouseUp

on initialise
  // Initialise the variables to null or 0
  put "" into name_of_person
  put 0 into first_number
  put 0 into second_number
  put 0 into my_answer
end initialise

on get_users_name @name_of_person
  // Setup a local variable to check if a key has been pressed
  local key_pressed

  // A loop is used to ask the user if they are happy with the name they have
  // entered. The loop expects either Y or y to be keyed in
  repeat until key_pressed = "Y" or key_pressed = "y"
    ask "Please enter your name"
    // If the cancel button is pressed, go back to the start of the program.
    if the result = "Cancel" then exit to top
    put it into name_of_person
    ask "Are you happy with the name entered? (Y or y for yes)"
    if the result = "Cancel" then exit to top
    put it into key_pressed
  end repeat
end get_users_name
```

The code is continued on the next page

Task 3: Three Additions

Implementation (continued)

```
on three_additions name_of_person, @first_number, @second_number, @my_answer
  //  Setup a local variable to keep track of the number of sums correct
  local number_correct
  put 0 into number_correct

  //  This fixed loop asks the user three basic arithmetic questions
  repeat with loop = 1 to 3
    //  The numbers are randomly generated between 1 and 9
    put random (9) into first_number
    put random (9) into second_number
    ask "What is"  &&first_number&& "added to" &&second_number&"?"
    if the result = "Cancel" then exit to top

    put it into my_answer

    put "So you think" &&first_number&& "added to" &&second_number&& "is"
    &&my_answer into line loop of field "output"  //  On the same line

    IF my_answer = first_number + second_number THEN
      answer "Correct answer, well done!"
      add 1 to number_correct
    ELSE
      answer "Wrong answer!"
    END IF
  end repeat

  put "Well" &&name_of_person& ", out of 3 you got" &&number_correct&& "correct."
  into line 5 of field "output"  //  On the same line

  If number_correct = 0 THEN put "You're not really good at basic arithmetic, you
  must practice more!" into line 7 of field "output"  //  On the same line
  If number_correct = 1 THEN put "You must practice more!  Try to improve on your
  mark for next time!" into line 7 of field "output"  //  On the same line
  If number_correct = 2 THEN put "Good effort!  Keep up the good work and try to
  get full marks next time!" into line 7 of field "output"  //  On the same line
  If number_correct = 3 THEN put "Full marks!  Well done!" into line 7 of field
  "output"  //  On the same line
end three_additions
```

Testing

Check that the program works correctly by keying in some test data. See if the **same results** are produced if you work out the **same answers** in your **head**.

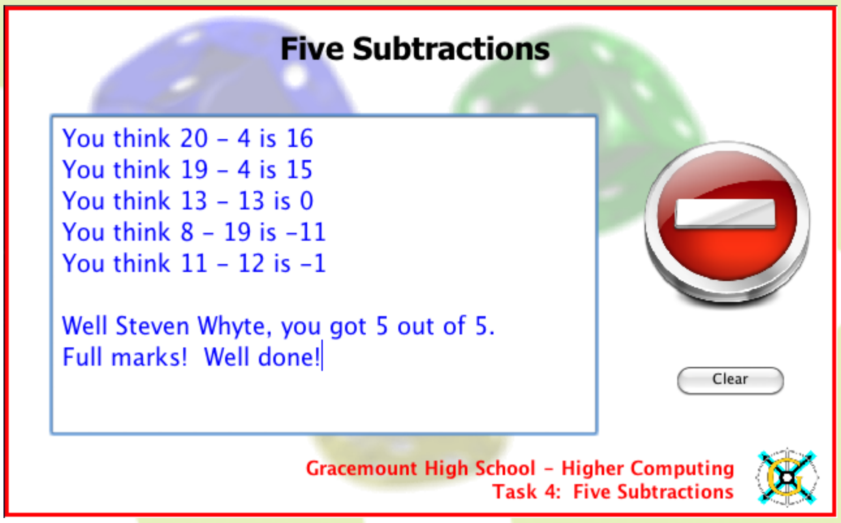
Task 4: Five Subtractions

Task

A program is required to test a users basic subtraction. The program will ask the user for their name, check to see if they are happy with the name entered and then ask them to answer **five subtractions** of two numbers (between **1** and **20**). Each answer will then be checked by the program and a comment about the answer will be displayed.

The **number correct** out of **five** along with a suitable comment **depending** on the **mark** they get should be displayed in the **output** field. You can make up whatever comments you like.

Sample output is shown below:


Five Subtractions
Clear
Gracemount High School – Higher Computing
Task 4: Five Subtractions

```
You think 20 - 4 is 16
You think 19 - 4 is 15
You think 13 - 13 is 0
You think 8 - 19 is -11
You think 11 - 12 is -1

Well Steven Whyte, you got 5 out of 5.
Full marks! Well done!
```

Your task is to do the following:

- Create the program script for the above problem using the code from the previous task to help (you may wish to **copy** the script and amend it).
- You **must** include **internal commentary** and **parameter passing**.
- Assign the code to the "**minus**" graphic.
- Create a **clear** button and **produce** a **simple script** to **clear** the **output** field.
  Note. *Look at the clear script from the previous program to help you.*
- **Test** that your program produces the **correct** results.
- Show the **teacher** your **working** program once **completed**.

The LiveCode stack to produce the script can be found on Glow:

LiveCode Programming Tasks > 4_Five Subtractions.livecode

Good Luck!

4...Five
Subtractions.livecode

Task 5: Choosing Colours

Specification

A **switch** statement can be very useful when you have a **number of possible inputs** and you want to respond **individually** to all the possibilities (**cases**).

For example: A program is required to take in a **colour** and **display** an appropriate **message** for that colour. The **card background** and **text** should also change to that colour.

If the colour entered **does not match** any of the **cases**, the program assumes the **output field is empty** and you should display a message to the user saying that there is no message for that colour.

Design: Pseudocode for the "Ask Colour" button

Stepwise Design (the main steps of the program)

```
1.      Choose Colour
```

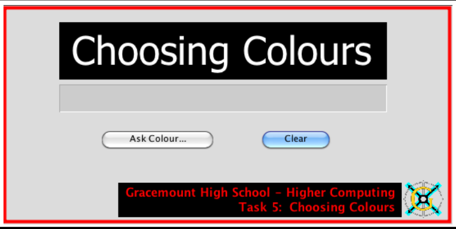
Stepwise Refinement (the main step further refined into smaller steps)

```
1.      Choose Colour
1.1       Setup theColour as a local variable
1.2       Ask the user for theColour
1.3       Put it into theColour

1.4       Start a Switch Control Structure using the variable theColour
1.5           In the case red is entered, change the background and font colour to red
1.6           Put a message saying blood is red into field "output"
1.7           Break out of switch statement

1.8           In the case blue is entered, set the background and font colour to blue
1.9           Put a message saying the sea is blue into field "output"
1.10          Break out of switch statement

1.11          In the case green is entered, set the background and font colour to green
1.12          Put a message saying grass is green into field "output"
1.13          Break out of switch statement

1.14          In the case black is entered, set the background and font colour to black
1.15          Put a message saying coal is black into field "output"
1.16          Break out of switch statement

1.17          In the case yellow is entered, set the background and font colour to yellow
1.18          Put a message saying the sun is yellow into field "output"
1.19          Break out of switch statement
1.20      End Switch

1.21      If the output field is empty then
1.22          Set the background colour to grey and text colour to black
1.23          Put a message saying that there is no message for that colour into field "output"
1.24      End If
```

Implementation

Open the "**Choosing Colors**" stack. It can be found on Glow:

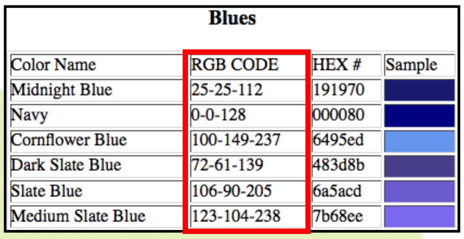
LiveCode Programming Tasks > 5_Choosing Colours.livecode



Task 5: Choosing Colours

Implementation

Add the following script to the **Ask Colour** button and **test** that your program produces the correct results for **each colour**. After you have finished, add another **three colours** of your choice along with your own message for each colour. Use the website www.tayloredmktg.com/rgb/ to get the RGB colour code for the colours you have chosen.



Blues

Color Name	RGB CODE	HEX #	Sample
Midnight Blue	25-25-112	191970	🟦
Navy	0-0-128	000080	🟦
Cornflower Blue	100-149-237	6495ed	🟦
Dark Slate Blue	72-61-139	483d8b	🟦
Slate Blue	106-90-205	6a5acd	🟦
Medium Slate Blue	123-104-238	7b68ee	🟦

```
on mouseUp
  choose_colour
end mouseUp

on choose_colour
  //  Setup the local variable to be used in this subroutine
  local theColour

  //  Setup the card
  put empty into field "output"
  set the backgroundColor of this card to 220,220,220
  set the textColor of field "output" to 0,0,0

  //  Prompt the user for their colour
  ask "Please enter your colour"
  put it into theColour

  //  Start a switch statement
  switch theColour
    //  In the case that the colour is red, blue, green, black or yellow, display a message and change
    //  the colour of the text and background to that colour using its RGB code.
    case "Red"
      put "Blood is red" into line 1 of field "output"
      set the backgroundColor of this card to 255,0,0
      set the textColor of field "output" to  255,0,0
      break
    case "Blue"
      put "The sea is blue" into line 1 of field "output"
      set the backgroundColor of this card to 0,0,255
      set the textColor of field "output" to  0,0,255
    case "Green"
      put "Grass is green" into line 1 of field "output"
      set the backgroundColor of this card to 85,107,47
      set the textColor of field "output" to  85,107,47
      break
    case "Black"
      put "Coal is black" into line 1 of field "output"
      set the backgroundColor of this card to 0,0,0
      set the textColor of field "output" to 0,0,0
      break
    case "Yellow"
      put "The sun is yellow" into line 1 of field "output"
      set the backgroundColor of this card to 255,255,0
      set the textColor of field "output" to 255,255,0
      break
  end switch

  //  If the user enters a colour not on the list above then set the colour to grey and text to black
  //  Display the following error message
  if field "output" is empty then
    set the backgroundColor of this card to 220,220,220
    set the textColor of field "output" to 0,0,0
    put "There is no message for that colour" into line 1 of field "output"
  end if
end choose_colour
```

Arrays

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

An **array** is a **structured data type** that is used for storing **sets of data within a single variable.**

To put it simply, an array is a **variable** which can store **more than one** piece of data in it so long as it is of the same **data type**.

Like variables, arrays must be **setup** at the start of an event. Look at and understand the example program below. It uses an array called **arrayname** and one variable called **maxstudents** which sets the **number** of **student names** to be **stored** in the **array** to 5.

```
//  Setup the global array and variable to be used in this event
global  arrayName , maxstudents

 on mouseUp
   //  Maxstudents will be set to five so five names will be entered and stored in the array
   put 5 into maxstudents
   get_student_name
   display_student_name
 end mouseUp

 on get_student_name
   //  Start a fixed loop which will repeat five times
   //  for each name to be stored in the arrayname
   repeat with loop = 1 to maxstudents
     //  Get the students name
     ask "Please enter the name of student: " & loop
     put it into arrayName[Loop]
   end repeat
 end get_student_name

 on display_student_name
   //  Start a fixed loop
   repeat with loop = 1 to maxstudents
     //  Put each name entered into arrayname into each
     //  line of the output field using loop
     put arrayName[Loop] into line loop of field "output"
   end repeat
 end display_student_name
```

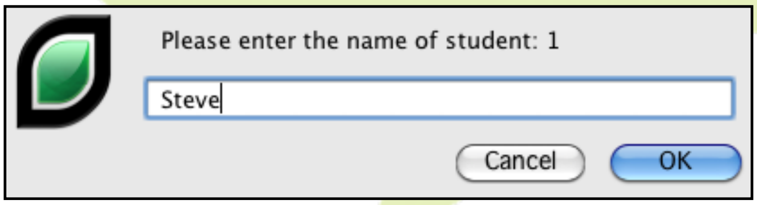
The program knows that **arrayname** is an **array** because of the **[loop]** straight after it.

[loop] indicates the current **element** (space) allocated to the array when it is used in a **loop**. This can be used to store the **users data**. In this program, the loop repeats **five times** as **maxstudents** is set to **5 in advance**.

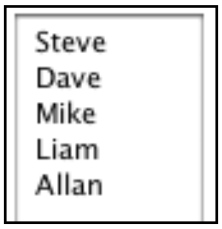
So, when the loop **starts**, the user can enter the five names, similar to that below. *Notice that all data in the array are of the same **type** in this case, **string** (text)*:

Repeat with loop = 1 to **maxstudents**
....
arrayname[loop] - "*Steve*" - **1st** pass of loop
arrayname[loop] - "*Dave*" - **2nd** pass of loop
arrayname[loop] - "*Mike*" - **3rd** pass of loop
arrayname[loop] - "*Liam*" - **4th** pass of loop
arrayname[loop] - "*Allan*" - **5th** pass of loop
End Repeat

The contents of the array can be displayed using the variable **loop** to ensure all values in each element (space) are displayed in each line (**loop**) 1, 2, 3, 4, 5 of the **output field**.


Please enter the name of student: 1
Steve

After the **five** names have been entered, the following output will be produced using **arrayname**:

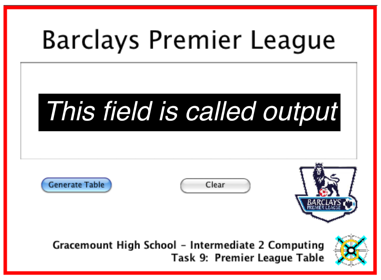

```
Steve
Dave
Mike
Liam
Allan
```

Arrays are a bit like bunk beds

Task 6: Premier League Table

Specification

A program is required to display the first **five** teams of the Barclay's Premier League. Separate arrays should be used to store the **name** of the team, **games played**, **goal difference (gd)** and amount of **points** achieved. This information should be displayed in an output field.


Barclays Premier League
This field is called output
Generate Table
Clear
Gracemount High School – Intermediate 2 Computing
Task 9: Premier League Table

Design: Pseudocode for "Generate Table" Button

Stepwise Design *(the main steps of the program)*

```
1.   Setup the Global arrays
2.   Initialise
3.   Read Data          In/Out: arrayname, arrayplayed, arraygd, arraypoints
4.   Display Data       Out:    arrayname, arrayplayed, arraygd, arraypoints
```

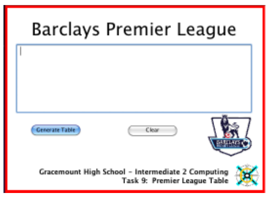
Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.   Setup the Global Arrays
1.1    Setup arrayname, arrayplayed, arraygd, arraypoints as the global arrays to be used in
       this event

2.   Initialise
2.1    Put "" into arrayname
2.2    Put 0 into arrayplayed, arraygd, arraypoints

3.   Read Data
3.1    Put "Manchster United", "Arsenal", "Manchester City", "Tottenham" and "Chelsea" into
       arrayname
3.2    Split arrayname by using a comma
3.3    Put 26, 26, 27, 26 and 26 into arrayplayed
3.4    Split arrayplayed by using a comma
3.5    Put 32, 29, 19, 9 and 24 into arraygd
3.6    Split arraygd by using a comma
3.7    Put 57, 53, 49, 47, 45 into arraypoints
3.8    Split arraygd by using a comma

4.   Display Data
4.1    Display the field headings using the tab function in line 1 of the field "output"
4.2    Display a dotted line under the headings in line 2 of the field "output"
4.3    Start a Repeat with loop 1 to 5
4.4       Put arrayname [loop]& tab & arrayplayed [loop] & tab & arraygd [loop] & tab &
          arraypoints [loop] into line loop+2 of field "output"
4.5    End Repeat
```

Implementation

Open the "**Premier League Table**" stack. It can be found on Glow:

LiveCode Programming Tasks > 6_Premier League Table.livecode


Barclays Premier League
Generate Table
Clear
Gracemount High School – Intermediate 2 Computing
Task 9: Premier League Table

Task 6: Premier League Table

Implementation

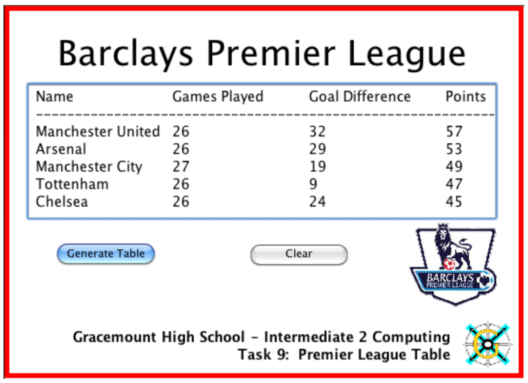
Assign the following script to the "**Generate Table button**". The purpose of this section of code is to copy the contents of the arrays into the output field at predefined points using the **tab** function.

```
// Setup the global arrays to be used in this event
global arrayname, arrayplayed,arraygd, arraypoints

on mouseUp
  initialise
  read_data arrayname, arrayplayed, arraygd, arraypoints
  display_data arrayname, arrayplayed, arraygd, arraypoints
end mouseUp

on initialise
  put "" into arrayname
  put 0 into arrayplayed
  put 0 into arraygd
  put 0 into arraypoints
end initialise

on read_data @arrayname, @arrayplayed, @arraygd, @arraypoints
  put "Manchester United","Arsenal","Manchester City","Tottenham","Chelsea" into arrayname
  split arrayname by comma
  put 26,26,27,26,26 into arrayplayed
  split arrayplayed by comma
  put 32,29,19,9,24 into arraygd
  split arraygd by comma
  put 57,53,49,47,45 into arraypoints
  split arraypoints by comma
end read_data

on display_data arrayname, arrayplayed, arraygd, arraypoints
   put "Name" & tab  & "Games Played" & tab & "Goal Difference" & tab & "Points" into line 1 of
   field "output"  //  on same line
   put "------------------------------------------------------------------------
   ----------------" into line 2 of field "output"  // on same line (88 –'s)
   repeat with loop = 1 to 5
      put arrayname[loop] & tab & arrayplayed[loop] & tab & arraygd[loop] & tab & arraypoints
      [loop] into line loop+2 of field "output"  //  on same line
   end repeat
end display_data
```

Testing

Test that the program produces the correct results from the four arrays as shown on the screenshot to the right:

Notice that the use of the **tab function** neatly lays out the data from the arrays into columns.

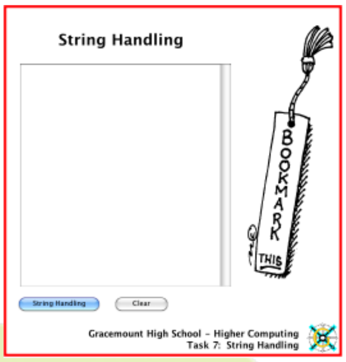


Barclays Premier League

```
Name              Games Played    Goal Difference    Points
-----------------------------------------------------------
Manchester United 26              32                 57
Arsenal           26              29                 53
Manchester City   27              19                 49
Tottenham         26              9                  47
Chelsea           26              24                 45
```

Gracemount High School – Intermediate 2 Computing
Task 9: Premier League Table

Task 7: String Handling



As well as handling numbers, LiveCode can also perform operations on **text**, also known as **string** variables.

The process of joining **two** or more **strings together** is called **concatenation**. This process is very useful if a program is perhaps required to generate a random **username** or **code** based on a certain number of **characters** contained within a users **forename** and **surname**.

Open the "**String Handling**" stack. It can be found in:

LiveCode Programming Tasks > 7_String Handling.livecode

Work through **each task** **one** by **one** and **after** completing **each** task, **run** your program to check that your program's output matches the **expected output**. You **don't** need to include the **internal commentary** but it's important that you **understand** how each section of code works as your practical coursework may require you to make use of string handling.

Add the following script to the "**String Handling**" button:

```
on mouseUp
  string_handling
end mouseUp

on string_handling
  //  Setup the local variables
  local first_word, second_word, complete_word, alphabet
  //
  //
  // ----------------------------------------
  //
  //
  //  Task 1
  //  Joining string variables together.  This process is called concatenation.
  put "book" into first_word
  put "mark" into second_word
  put first_word into complete_word
  put second_word after complete_word
  put complete_word into line 1 of field "output"
  //  RUN YOUR PROGRAM NOW....
  // ----------------------------------------
  //
  //
  //  Task 2
  //  Create the text to go into the string variable alphabet
  put "abcdefghijklmnopqrstuvwxyz" into alphabet
  //
  // ----------------------------------------
  //
  //
  //  Task 2 (a)
  //  Put character 3 of the string variable alphabet into line 3 of field output
  put char 3 of alphabet into line 3 of field "output"
  //  RUN YOUR PROGRAM NOW....
  //
  //  More tasks over the page.
```

The following output should be produced:

```
bookmark
```

The following output should be produced:

```
c
```

The String Handling code is continued on the next page

Task 7: String Handling (continued)

```
// ----------------------------------------
//
// Task 2 (b)
// Put characters 1 to 3 of the string variable alphabet into line 5 of field output
put char 1 to 3 of alphabet into line 5 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

abc

```
//----------------------------------------
//
// Task 2 (c)
// Put characters 24 to 26 of the string variable alphabet into line 7 of field output
put char 24 to 26 of alphabet into line 7 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

xyz

```
//----------------------------------------
//
//
// Task 2 (d)
// Put characters 10 to 20 of the string variable alphabet into line 9 of field output
put char 10 to 20 of alphabet into line 9 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

jklmnopqrst

```
//----------------------------------------
//
//
// Task 2 (e)
// Put the length of the string variable alphabet into line 11 of field output
put the length of alphabet into line 11 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

26

```
//----------------------------------------
//
//
// Task 2 (f)
// Find the position of a character in a string variable
// This example finds "c" in the string variable alphabet to produce the value of 3
put offset("c",alphabet) into line 13 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

3

```
//----------------------------------------
//
//
// Task 3: Number to Character upper case
// Produces a random upper case value from A – Z
// Capital "A" starts at ASCII 65 + 25 other characters of alphabet
put NumToChar (random(26) + 64) into line 15 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

any UPPER case (A-Z)

```
//----------------------------------------
//
//
// Task 4: Number to Character lower case
// Produces a random lower case value from a – z
// Lower case "a" starts at ASCII 96 + 25 other characters of alphabet
put NumToChar (random(26) + 95) into line 17 of field “output”
// RUN YOUR PROGRAM NOW....
```

The following output should be produced:

any lower case (a-z)

```
//----------------------------------------
//
//
// Task  5: Character to Number
// Produces the ASCII code value for the chosen character
put CharToNum ("A") into line 19 of field “output”
// RUN YOUR PROGRAM NOW....
end string_handling
```

The following output should be produced:

65

Task 8: Customer Code Generator

Task

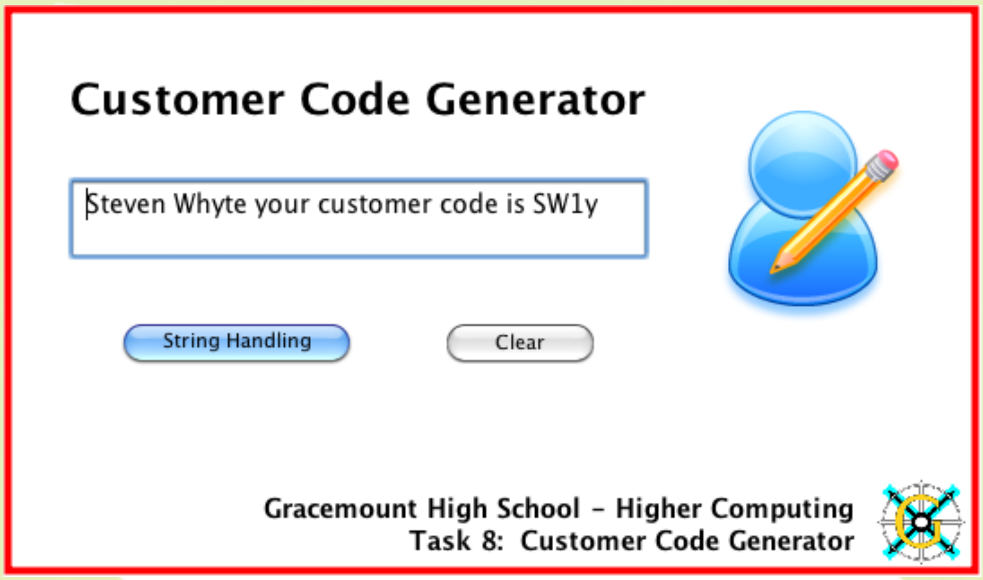
A program is required to generate a customer code. The program should **Ask** the user for their **first name** and **second name**.

Once the program has this information, the customer code should be generated.

The customer code should be made up of:

- The **first character** from both the **first name** and **second name**.
- A **random number** number between **1** and **9**.
- A **random lower** case **character** (a-z).

Sample output is shown below assuming the name of **Steven Whyte** has been entered:



Customer Code Generator

```
Steven Whyte your customer code is SW1y
```

String Handling
Clear
Gracemount High School - Higher Computing
Task 8: Customer Code Generator

Your task is to do the following:

- Produce the program code for this solution using the string handling examples on the previous pages.
- Create a **clear** button and **produce** a simple **script** to **clear** the **output** field.
- **Test** that your solution works by correctly producing the **customer code** as shown above.
- Show the **teacher** your **working** program once **completed**.

Your code should be placed into the **"Generate Code"** button of the "**Customer Code Generator**" stack and the output should be displayed in the **output** field.

The "**Customer Code Generator**" stack can be found on Glow:

LiveCode Programming Tasks > 8_Customer Code Generator.livecode

8...Customer Code Generator.livecode

Good Luck!

Task 9: Student Marks

Specification

A program is required by a teacher to allow her to store **marks** and **grades** for her Higher Computing class. You have been asked to produce a sample program to store the details of **five** students.

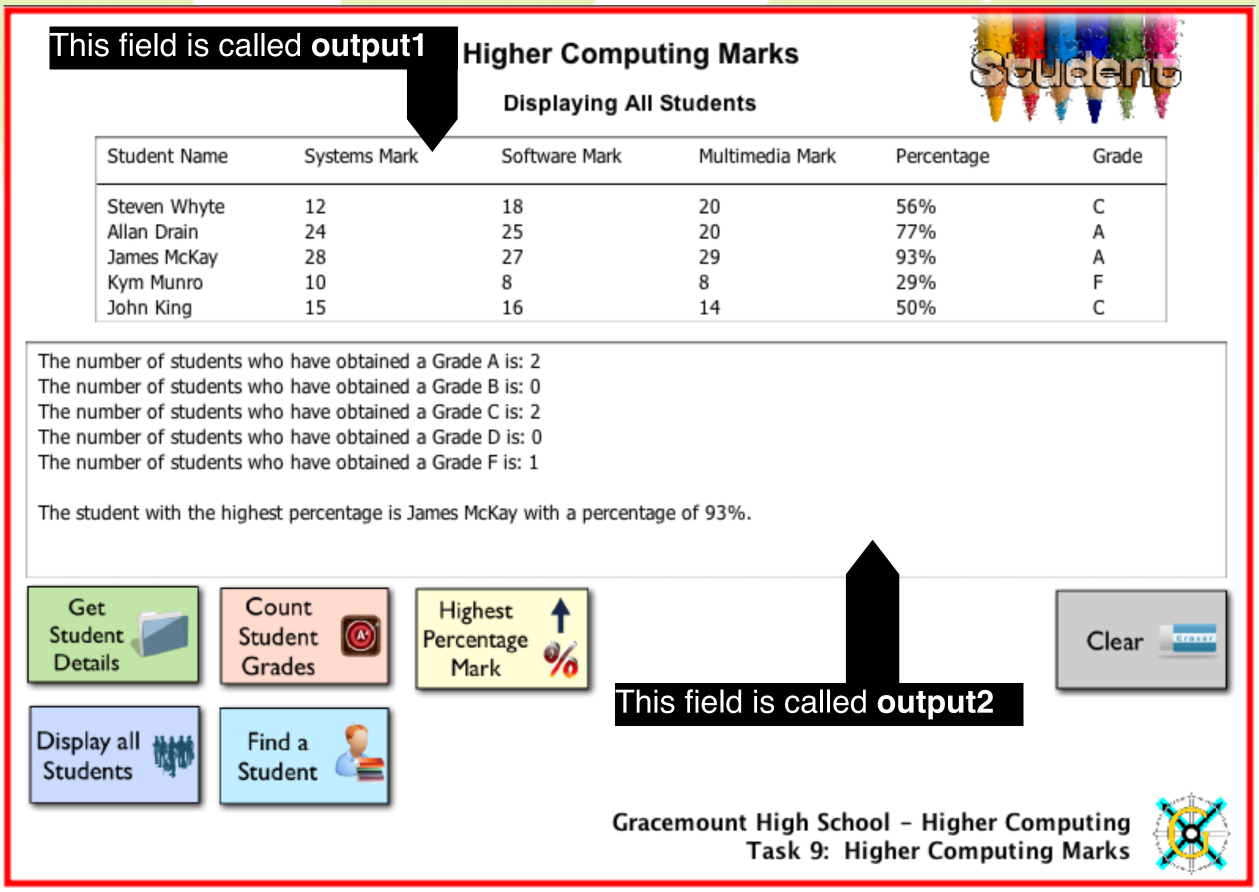
This program should allow the teacher to get the **names** and **marks** of the three main topics studied at Higher level. Each mark should be **validated** as **whole numbers** between **0** and **30** *(input validation algorithm)*.

Once these details have been keyed in, the program should work out the **percentage mark** and **final grade** based on the student's **percentage** mark. All of these details should then be displayed appropriately in a field called **output1**.

The program should also allow the teacher to:

- **Count** the **occurrences** of each **grade** A, B, C, D and F *(count occurrences algorithm)*.
- **Search** on a **student name** *(linear search algorithm)*.
- **Find** the **student** with the **highest percentage** *(find maximum algorithm)*.

These details will be displayed in a field called **output2**. Sample output from the program is shown below. You may wish to use the **same** test data when it comes to **testing** your program.



```
Student Name    Systems Mark    Software Mark    Multimedia Mark    Percentage    Grade

Steven Whyte    12              18               20                 56%           C
Allan Drain     24              25               20                 77%           A
James McKay     28              27               29                 93%           A
Kym Munro       10              8                8                  29%           F
John King       15              16               14                 50%           C
```

```
The number of students who have obtained a Grade A is: 2
The number of students who have obtained a Grade B is: 0
The number of students who have obtained a Grade C is: 2
The number of students who have obtained a Grade D is: 0
The number of students who have obtained a Grade F is: 1

The student with the highest percentage is James McKay with a percentage of 93%.
```

Read through the **design** of **Get Student Details** over the page to understand what is involved and then key in the script for this event. The script is supplied for you on pages **29** and **30**.

Task 9: Student Marks

Design for “Get Student Details” Graphic

Stepwise Design *(the main steps of the program with data flow)*

1.	Setup	
2.	Get Student Details	**In**: maxstudents **In/Out**: arrayname, arraycs, arraysdp, arraymm, check_number
3.	Calculate Percentage	**In**: maxstudents, arraycs, arraysdp, arraymtm **In/Out**: arraypercentage
4.	Get Grade	**In**: maxstudents, arraypercentage **In/Out**: arraygrade
5.	Display Details	**In**: maxstudents, arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade
6.	Validate	**In**: check_number

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup
1.1     Setup arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade as global arrays
1.2     Setup maxstudents and check_number as global variables
1.3     Put the value of 5 into the variable maxstudents
1.4     Clear the “heading”, “output1” and “output2” fields
1.5     Put the text “Enter Student Details” into the field “heading”

2.    Get Student Details
2.1     Start a Repeat with loop = 1 to maxstudents
2.2       Ask for the students name
2.3       If the user selects the cancel button then exit to the top of the program
2.4       Put it into arrayname[loop]

2.5       Ask for the students Computer Systems Mark (0-30)
2.6       If the user selects the cancel button then exit to the top of the program
2.7       Put it into check_number
2.8       Call the validation function
2.9       Put check_number into arraycs[loop]

2.10      Ask for the students Software Development Mark (0-30)
2.11      If the user selects the cancel button then exit to the top of the program
2.12      Put it into check_number
2.13      Call the validation function
2.14      Put check_number into arraysdp[loop]

2.15      Ask for the students Multimedia Mark (0-30)
2.16      If the user selects the cancel button then exit to the top of the program
2.17      Put it into check_number
2.18      Call the validation function
2.19      Put check_number into arraymm[loop]
2.20    End Repeat
```

The design is continued on the next page

Task 9: Student Marks

Design for “Get Student Details” Graphic (continued)

```
3.    Calculate Percentage
3.1     Start a Repeat with loop = 1 to maxstudents
3.2     Put (arraycs[loop] + arraysdp[loop] + arraymm[loop]) / by 90 * by 100  into arraypercentage[loop]
3.3     End Repeat

4.    Get Grade
4.1     Start a Repeat with loop = 1 to maxstudents
4.2        If the arraypercentage[loop] is greater than 70 then put “A” into arraygrade[loop]
4.3        If the arraypercentage[loop] is between 60 and 69 then put “B” into arraygrade[loop]
4.4        If the arraypercentage[loop] is between 50 and 59 then put “C” into arraygrade[loop]
4.5        If the arraypercentag [loop] is between 40 and 49 then put “D” into arraygrade[loop]
4.6        If the arraypercentage[loop] is less than 40 then put “F” into arraygrade[loop]
4.7     End Repeat

5.    Display Details
5.1     Set the number format to 0
5.2     Put the column headings of “Student Name” tab, “Systems Mark”, tab, “Software Mark”, tab,
        “Multimedia Mark”, tab, “Percentage”, tab, “Grade” into line 1 of field “output1”
5.3     Start a Repeat with loop 1 to maxstudents
5.4       Put arrayname[loop] & tab & arrayc[loop] & tab & arraysd [loop] & tab & arraymm[loop] & tab
          & arraypercentage[loop] & tab & arraygrade[loop] into line loop+3 of field “output1”
5.5     End Repeat

6.    Validate
6.1     Start a Repeat until check_number is between 0 and 30 and is an integer
6.2        Ask the user to re-enter the mark if it is invalid
6.3        If the user selects the cancel button then exit to the top of the program
6.4        Put it into check_number
6.5     End Repeat
```

Input Validation Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the first script.

After carefully reading through the design. You should begin to code the script for the first button called “Get Student Details”. Key in all of the code over the page **carefully**.

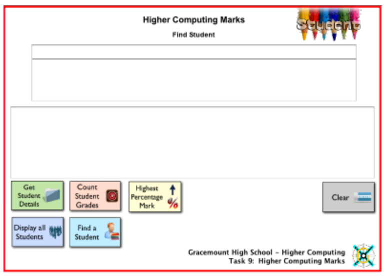
Note. You **do not** need to include **internal commentary** at this point. The commentary is only there to help you understand what is going on.

You will however need to produce internal commentary when it comes to completing your SQA coursework.

After completing **each** event, you should **test** that your program is **working correctly** using the supplied **test data**.

Open the “**Higher Computing Marks**” stack. It can be found on Glow:

LiveCode Programming Tasks > 9_Higher Computing Marks.livecode



Task 9: Student Marks

Implementation: Script “Get Student Details”

Key the following code into the **Get Student Details** button. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Setup the global arrays and variables to be used in this event
global arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade, maxstudents, check_number

// When the mouse up event is detected on this button, execute following subroutines
on mouseUp
  set_up maxstudents
  get_student_details maxstudents, arrayname, check_number, arraycs, arraysdp, arraymm
  calculate_percentage maxstudents, arraycs, arraysdp, arraymm, arraypercentage
  get_grade maxstudents, arraypercentage, arraygrade
  display_details maxstudents, arraycs, arraysdp, arraymm, arraypercentage, arraygrade
  validate check_number
end mouseUp

on set_up @maxstudents
  put 5 into maxstudents  //  5 students maximum
  put empty into field "output1"  //  Clear the fields
  put empty into field "output2"
  put empty into field "heading"
  put "Enter Student Details" into field "heading"  // Display heading "Enter Student Details"
end set_up

on get_student_details maxstudents, @arrayname, @check_number, @arraycs, @arraysdp, @arraymm
    repeat with loop = 1 to maxstudents //  Loop for 1 to 5 students

    //  Get the students name
    ask "Please enter the name of student: " & loop
    if the result = "Cancel" then exit to top //  If the cancel button is pressed, exit to top
    put it into arrayname[Loop]

    //  Get the students validated Computer Systems mark
    ask "Please enter " & arrayname[Loop] & "'s mark for Computer Systems out of 30: "
    if the result = "Cancel" then exit to top
    put it into check_number
    validate  //  Start the validation function at the bottom of the event
    Put check_number into arraycs[Loop]  //  Put the validated number into the array

    //  Get the students validated Software Development mark
    ask "Please enter " & arrayname[Loop] & "'s mark for Software Development out of 30: "
    if the result = "Cancel" then exit to top
    put it into check_number
    validate  //  Start the validation function at the bottom of the event
    put check_number into arraysdp[Loop]  //  Put the validated number into the array

    //  Get the students validated Multimedia Technology mark
    ask "Please enter " & arrayname[Loop] & "'s mark for Multimedia out of 30: "
    if the result = "Cancel" then exit to top
    put it into check_number
    validate  //  Start the validation function at the bottom of the event
    put check_number into arraymm[Loop]  //  Put the validated number into the array
  end repeat
end get_student_details
```

The code is continued on the next page

Task 9: Student Marks

```
on calculate_percentage maxstudents, arraycs, arraysdp, arraymt, @arraypercentage
  repeat with loop = 1 to maxstudents  //  Loop for 1 to five students
    //  Calculate the students percentage mark out of the three tests
    put (arraycs[Loop] + arraysdp[Loop] + arraymm[Loop]) / 90 * 100 into arraypercentage[loop]
  end repeat
end calculate_percentage

on get_grade maxstudents, arraypercentage, @arraygrade
  //  Use of multiple IF's to determine what grade a student gets based on their overall percentage
  repeat with loop = 1 to maxstudents  //  Loop for 1 to 5 students
    IF arraypercentage[loop] >= 70 then put "A" into arraygrade[loop]
    IF arraypercentage[loop] >= 60 AND arraypercentage[loop] <= 69 then put "B" into arraygrade[loop]
    IF arraypercentage[loop] >= 50 AND arraypercentage[loop] <= 59 then put "C" into arraygrade[loop]
    IF arraypercentage[loop] >= 40 AND arraypercentage[loop] <= 49 then put "D" into arraygrade[loop]
    IF arraypercentage[loop] < 40 then put "F" into arraygrade[loop]
  end repeat
end get_grade

on display_details maxstudents, arraycs, arraysdp, arraymm, arraypercentage, arraygrade

  set numberformat to "0"  // Set the format of any numbers displayed to 0
  //  Display the headings
  put "Student Name" & tab  & "Systems Mark" & tab & "Software Mark" & tab & "Multimedia Mark" & tab &
       "Percentage" & tab & "Grade" into line 1 of field "output1" //  On the same line

  repeat with loop = 1 to maxstudents  //  Loop for 1 to 5 students
    //  Put the array data into field "output1"
    put arrayname[loop] & tab & arraycs[loop] & tab & arraysdp[loop] & tab & arraymm[loop] & tab &
    arraypercentage[loop] &"%" & tab & arraygrade[loop] into line loop+2 of field "output1"  //  Same line
  end repeat

  disable image "buttonGetStudents" //  Once all details are displayed, disable this button.
                                    //  It will be enabled when the user selects the clear button
end display_details

on validate check_number
  //  ***Input Validation***
  //  The validation function will ensure the user has entered a whole number between 1 and 30
  repeat until check_number >= 0 and check_number <= 30 and check_number is an integer
    ask "You have entered an invalid guess, please enter a whole number between 0 and 30."
    if the result = "Cancel" then exit to top
    put it into check_number
  end repeat
end validate
```

Testing

You should now test that your program is working correctly. Key in the following **names** and **marks** for the **three** assessments below. Check that your **percentage mark** and **grade** is the **same** as below.

If they are the same then your **percentage** and **grade** have been **calculated correctly** and your program **works**.

Student Name	Systems Mark	Software Mark	Multimedia Mark	Percentage	Grade
Steven Whyte	30	20	10	67%	B
Allan Drain	30	28	25	92%	A
David Beckham	10	5	15	33%	F
Edward Smith	17	19	18	60%	B
Lisa Smyth	28	27	30	94%	A

Task 9: Student Marks

Design for "Count Student Grades" Graphic

Stepwise Design *(the main steps of the program with data flow)*

```
1.   Setup
2.   Count Occurrences          In: maxstudents, arraygrade
```

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup
1.1     Pass in maxstudents as the global variable to be used in this event
1.2     Pass in arraygrade and as the global array to be used in this event
1.3     Clear the field "heading"
1.4     Put the text "Count Student Grades" into the field "heading"

2.    Count Occurrences
2.1     Setup, AGrade, BGrade, CGrade, DGrade and FGrade as local variables
2.2     Put 0 into all of the local variables
2.3     Start a Repeat with loop 1 to maxstudents
2.4        If the arraygrade[loop] is equal to an "A" then add 1 to AGrade
2.5        If the arraygrade[loop] is equal to an "B" then add 1 to BGrade
2.6        If the arraygrade[loop] is equal to an "C" then add 1 to CGrade
2.7        If the arraygrade[loop] is equal to an "D" then add 1 to DGrade
2.8        If the arraygrade[loop] is equal to an "F" then add 1 to FGrade
2.9     End Repeat

2.10    Put a message showing the number of pupils with an A into line 1 of field "output2" using AGrade
2.11    Put a message showing the number of pupils with a B into line 2 of field "output2" using BGrade
2.12    Put a message showing the number of pupils with a C into line 3 of field "output2" using CGrade
2.13    Put a message showing the number of pupils with a D into line 4 of field "output2" using DGrade
2.14    Put a message showing the number of pupils with an F into line 5 of field "output2" using FGrade
```

Counting Occurrences Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the second script.

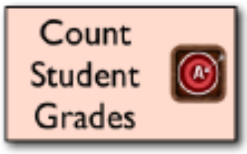
After carefully reading through the design above. You should begin to code the script for the second button called "**Count Student Grades**". Key in all of the code on the next page **carefully** and correct your errors.

This event will count the number of **A**, **B**, **C**, **D** and **F** grades in the sample class of five students using the **arraygrade**. Each count of grade will then be placed into separate local variables of **AGrade**, **BGrade**, **CGrade**, **DGrade** and **FGrade** as shown above.

After completing the script, you should **test** that your program is working **correctly** by producing the **occurrence** of **each grade obtained**. The predicted test data is shown at the **bottom** of the **next** page.

Task 9: Student Marks

Implementation: Script “Count Student Grades”

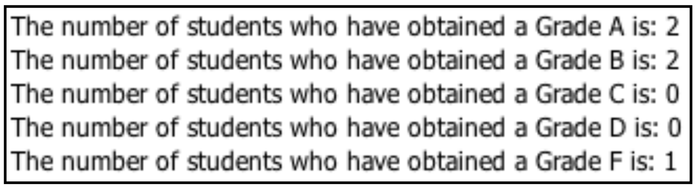


Key the following code into the **Count Student Grades** Button. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Allow access to global arrays and variables to be used in this event
global arraygrade, maxstudents

// When mouse up event is detected on this button then execute following subroutines
on mouseUp
  set_up
  count_occurrences maxstudents, arraygrade
end mouseUp

on set_up
  put empty into field "heading"  //  Clear the field
  put "Count Student Grades" into field "heading"   // Display the heading "Count Student Grades"
end set_up

on count_occurrences maxstudents, arraygrade
  local AGrade, BGrade, CGrade, DGrade, FGrade  // Setup local variables

  put 0 into AGrade  // zero the local variables
  put 0 into BGrade
  put 0 into CGrade
  put 0 into DGrade
  put 0 into FGrade

  //  ***Counting Occurrences***
  // Count the number of A, B, C, D and F grades obtained by the class
  repeat with loop = 1 to maxstudents  //  Loop for 1 to 5 students
    IF arraygrade[loop] = "A" then add 1 to AGrade
    IF arraygrade[loop] = "B" then add 1 to BGrade
    IF arraygrade[loop] = "C" then add 1 to CGrade
    IF arraygrade[loop] = "D" then add 1 to DGrade
    IF arraygrade[loop] = "F" then add 1 to FGrade
  end repeat

  //  Print the results into field “output2”
  put "The number of students who have obtained a Grade A is: " & AGrade into line 1 of field "output2"
  put "The number of students who have obtained a Grade B is: " & BGrade into line 2 of field "output2"
  put "The number of students who have obtained a Grade C is: " & CGrade into line 3 of field "output2"
  put "The number of students who have obtained a Grade D is: " & DGrade into line 4 of field "output2"
  put "The number of students who have obtained a Grade F is: " & FGrade into line 5 of field "output2"
end count_occurrences
```

Testing

Test that your program successfully counts the number of each occurrence of grade **A**, **B**, **C**, **D** and **F** and places this into the **output2** field. Your results should look the same as the results below if you are using the **same** test data as you keyed in on page **32**.



```
The number of students who have obtained a Grade A is: 2
The number of students who have obtained a Grade B is: 2
The number of students who have obtained a Grade C is: 0
The number of students who have obtained a Grade D is: 0
The number of students who have obtained a Grade F is: 1
```

Task 9: Student Marks

Design for "Find a Student" Graphic

Stepwise Design (the main steps of the program with data flow)

1. Setup
2. Find Student — **In**: maxstudents; **In**: arrayname, arraycs, arraysdp, arraymt, arraypercentage, arraygrade

Stepwise Refinement (the main steps further refined into smaller steps)

```
1.    Setup
1.1     Pass in the global variable maxstudents to be used in this event
1.2     Pass in the global arrays arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade
        to be used in this event
1.3     Clear the field "heading"
1.4     Put the text "Find Student" into the field "heading"

2.    Find Student
2.1     Setup student_name and found as local variables
2.2     Ask for the name of the student to find
2.3     Put it into student_name
2.4     Set the number format to 0
2.5     Put the column headings of "Student Name" & tab & "Systems Mark" & tab & "Software Mark" &
        tab & "Multimedia Mark" & tab & "Percentage" & tab & "Grade" into line 1 of field "output1"
2.6     Put false into the boolean variable found
2.7     Start a Repeat with loop = 1 to maxstudents
2.8       If the arrayname[loop] = student_name then
2.9         Put arrayname[loop] & tab & arraycs[loop] & tab & arraysdp[loop] & tab& arraymm[loop]
             tab & arraypercentage[loop] & tab & arraygrade[loop] into loop+2 of field "output1"
2.10        Put true into the boolean variable found
2.11      End If
2.12    End Repeat
2.13    If boolean variable found = false then
2.14      Put a message telling the user that no students found into loop+2 of field "output1"
2.15    End If
```

Linear Search Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the third script.

After carefully reading through the design above. You should begin to code the script for the third button called "**Find a Student**". Key in all of the code **carefully**.

This event will **compare** the users **search** with each name in **arrayname** and display the student's details in the **output1** field. If no name is found in arrayname then a message explaining that **no students** have been **found** is displayed. Notice that a **boolean** (true/false) variable is used to **determine** whether or not to display the **no students found** message.

After completing the script, you should **test** that your program is working correctly. The predicted test data is shown at the bottom of the next page assuming the search of "**Steven Whyte**" is entered.

Task 9: Student Marks

Implementation: Script “Find a Student”

Key the following code into the **Find a Student** button . You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Allow access to global arrays and variables to be used in this event
global maxstudents, arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade

// When mouse up event is detected on this button then execute following subroutines
on mouseUp
  set_up
  find_student arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade, maxstudents
end mouseUp

on set_up
  put empty into field "heading"  //  Clear the fields
  put empty into field "output1"
  put "Find Student" into field "heading"  // Display heading "Find Student" for this event
end set_up

on find_student arrayname, arraycs, arraysdp, arraymm, arraypercentage, arraygrade, maxstudents
  local student_name, found  // Setup local variables

  ask "Please enter the name of the student:"
  put it into student_name  //  User enters the name to search

  set numberformat to "0"  // Set the format of any numbers displayed to 0
  //  Display the headings
  put "Student Name" & tab  & "Systems Mark" & tab & "Software Mark" & tab & "Multimedia Mark" & tab &
       "Percentage" & tab & "Grade" into line 1 of field "output1"  //  on the same line
  put false into found  //  Set the found boolean variable to false

  //  ***Linear search***
  //  Search for a student based on the name the user has entered
  repeat with loop = 1 to maxstudents  //  Loop for 1 to 5 students
    if arrayname[Loop] = student_name then  //  If the names array is equal to the search name then..
      //  Put the array data into field “output1”
      put arrayname[loop] & tab & arraycs[loop] & tab & arraysdp[loop] & tab & arraymm[loop] & tab &
      arraypercentage[loop] &"%" & tab & arraygrade[loop] into line loop+2 of field "output1" //  on the
                                                                                              //  same line
      put true into found  //  if found set to true
    end if
  end repeat

  //  If no match found, set found to false and then print a suitable message into field “output1”
  if found = false then put "***************************** No students with that name have been found
  *****************************" into line 3 of field "output1"  //  on the same line (29 stars each side)
end find_student
```

Testing

Test that your program successfully **finds** a student once you **search** on their **name**. This should be displayed in the **output1** field.

Student Name	Systems Mark	Software Mark	Multimedia Mark	Percentage	Grade
Steven Whyte	30	20	10	67%	B

Task 9: Student Marks

Design for "Highest Percentage Mark" Graphic

Stepwise Design **(the main steps of the program with data flow)**

```
1.  Setup
2.  Find Maximum Percentage        In: maxstudents, arrayname, arraypercentage
```

Stepwise Refinement **(the main steps further refined into smaller steps)**

```
1.    Setup
1.1     Pass in maxstudents as the global variable to be used in this event
1.2     Pass in arrayname and arraypercentage as the global arrays to be used in this event
1.2     Clear the field "heading"
1.3     Put the text "Highest Student Percentage" into the field "heading"

2.    Find Maximum Percentage
2.1     Setup maximum and position as local variables
2.2     Put 0 into maximum
2.3     Start a Repeat with loop = 1 to maxstudents
2.4       If the arraypercentage[loop] is greater than maximum then
2.5         Put arraypercentage[loop] into maximum
2.6         Put loop into position
2.7       End If
2.8     End Repeat

2.9     Set the number format of any number shown to a whole number (0)
2.10    Put a message showing name and percentage of the student who obtained the highest
        percentage using the variable position into line 7 of field "output2"
```

Find Max Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the fourth script.

After carefully reading through the design above. You should begin to code the script for the fourth button called "**Highest Percentage Mark**". Key in all of the code **carefully**.

This event will **find** and **display** the **highest percentage** mark using the **arraypercentage**.

Once the highest percentage has been found, the program will produce the **name** and **percentage mark** of the student with the **highest percentage**.

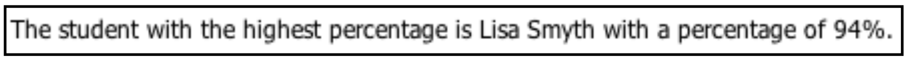
After completing the script, you should **test** that the program is working correctly. The predicted test data is shown at the **top** of the **next page**.

You should also key in the code to **clear** the **output** and **text fields**. This code is displayed at the bottom of the next page and should be assigned to the **clear** button.

Task 9: Student Marks

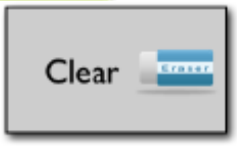
Implementation: Script “Highest Percentage Mark”

Key the following code into the **Highest Percentage Mark**. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on. Once completed, **test** that your program correctly identifies the **student** with the **highest percentage mark**. This should be placed into **line 7** of the **output2** field.


The student with the highest percentage is Lisa Smyth with a percentage of 94%.

```
// Allow access to global arrays and variables to be used in this event
global maxstudents, arrayname, arraypercentage

// When mouse up event is detected on this button then execute following subroutines
on mouseUp
  set_up
  find_maximum_percentage maxstudents, arrayname, arraypercentage
end mouseUp

on set_up
  put empty into field "heading"  //  Clear the fields
  put "Highest Student Percentage" into field "heading"  //  Display heading "Highest Student
                                                          //  Percentage" for this event
end set_up

on find_maximum_percentage maxstudents, arrayname, arraypercentage
  local maximum, position   //  Setup local variables
  put 0 into maximum        //  Zero the Maximum variable

  //  ***Find Maximum***
  // Find the student with the highest percentage
  Repeat with loop = 1 to maxstudents            //  Loop for 1 to 5 students
    If arraypercentage[loop] > maximum then      //  If the percentage array is greater than Maximum
      put arraypercentage[loop] into maximum     //  Put the value from the array into Maximum
      put loop into position                     //  Record the position of the loop
    end if
  end Repeat

  set numberformat to "0"  // Set the format of any numbers displayed to 0
  //  Display the position student with the highest percentage in field “Output2”
  put "The student with the highest percentage is " & arrayname[position] & " with a percentage of " &
      arraypercentage[position] & "%." into line 7 of field "output2"  //  On same line
end find_maximum_percentage
```

Implementation: Script “Clear”

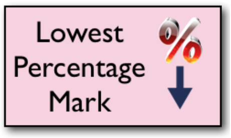


Once you have tested that your highest percentage button is working, key in the following code for the clear button **carefully**. You don’t need to include the internal commentary, it is only there to help you understand what is going on.

```
// When the mouse up event is detected on this button, execute following actions
on mouseUp
  put empty into field "output1"       //  Clear the fields
  put empty into field "output2"
  put empty into field "heading"
  enable image "buttonGetStudents"  //  Enable the image button “Get Students” once the clear button
                                    //  has been pressed
end mouseUp
```

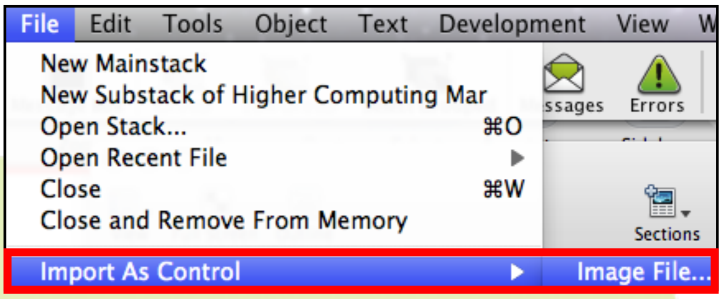
Task 9: Student Marks

Task



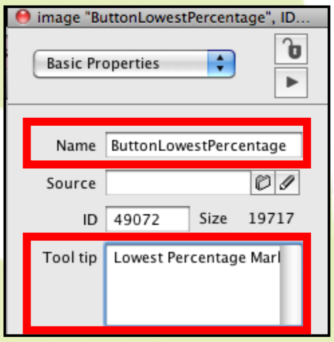
You are now going to attempt to create the script for **one** further button in the Student Marks program.

This button is going to find the **lowest percentage mark** and **display** the **name** and **percentage** mark into **line 8** of the **output2** field.



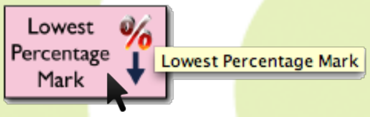
The script you will enter is very similar to the Highest Percentage Mark so use this to help you. Also use the **pseudocode** on the **next page** to help you.

Before you begin creating the **script**, you **must do** the following:



Name: ButtonLowestPercentage
ID: 49072
Tool tip: Lowest Percentage Mar

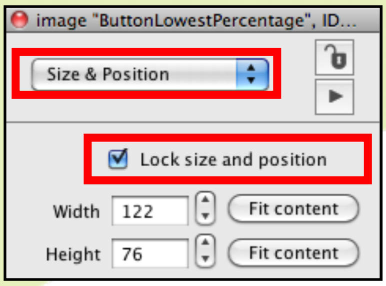
- Import the image "9B...Lowest Percentage Mark Graphic.png" by **importing** the graphic **as** a **control**, as shown above. It can be found in your **LiveCode Programming Tasks** folder.
- Double click on the imported image and give the image a **name** and a **tooltip** as shown on the right.



Note. *A tooltip is a small yellow hover box which appears with information in it once the cursor passes over an object as shown on the image above.*

- Once you have resized the graphic and moved it into its desired position, you **must lock** it's **size** and **position** to prevent it from going back to its normal size.

Note. *To do this, double click on the graphic to bring up the properties inspector again and select "size and position". Check the box beside "lock size and position" as shown below.*



☑ Lock size and position
Width: 122
Height: 76

You are now ready to create the script for the Lowest Percentage Mark. Make sure that you **test** that your script produces the **correct result** in the **output2** field once complete.

Remember, if you get stuck, **think!** You've already created the script to find the **highest percentage**. Therefore, the script to find **lowest** percentage is going to be **very similar**! Use the **pseudocode** over the page to help.

Good Luck!

Task 9: Student Marks

Design for "Lowest Percentage Mark" Graphic

Stepwise Design (the main steps of the program with data flow)

```
1.   Setup
2.   Find Minimum Percentage          In: maxstudents, arrayname, arraypercentage
```

Stepwise Refinement (the main steps further refined into smaller steps)

```
1.     Setup
1.1      Pass in maxstudents as the global variables to be used in this event
1.2      Pass in arrayname and arraypercentage as the global arrays to be used in this event
1.2      Clear the field "heading"
1.3      Put the text "Lowest Student Percentage" into the field "heading"

2.     Find Minimum Percentage
2.1      Setup minimum and position as local variables
2.2      Put 100 into minimum
2.3      Start a Repeat with loop 1 to maxstudents
2.4         If the arraypercentage[loop] is less than minimum then
2.5            Put arraypercentage[loop] into minimum
2.6            Put loop into position
2.7         End If
2.8      End Repeat

2.9      Set the number format of any number shown to a whole number (0)
2.10     Put a message showing name and percentage of the student who obtained the lowest
         percentage using the variable position into line 8 of field "output2"
```

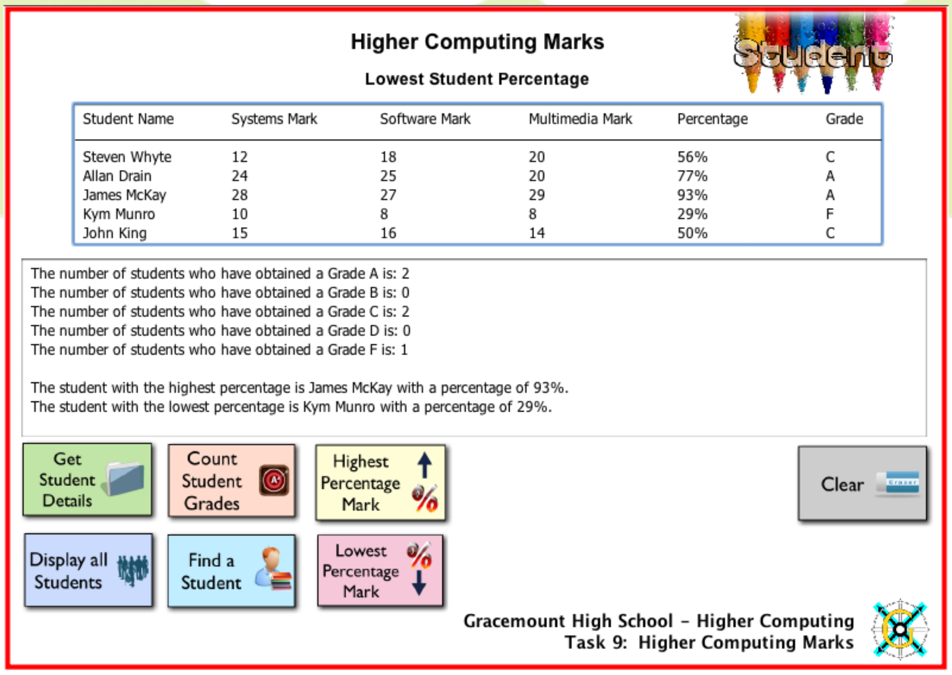
Find Min Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Testing

Test that your program correctly produces the student with the **lowest overall percentage** in **line 8** of the **output2** field. Show the **teacher** your **working** program once **completed**.

You may wish to use the **same test** data as shown below:



Higher Computing Marks

Lowest Student Percentage

Student Name	Systems Mark	Software Mark	Multimedia Mark	Percentage	Grade
Steven Whyte	12	18	20	56%	C
Allan Drain	24	25	20	77%	A
James McKay	28	27	29	93%	A
Kym Munro	10	8	8	29%	F
John King	15	16	14	50%	C

```
The number of students who have obtained a Grade A is: 2
The number of students who have obtained a Grade B is: 0
The number of students who have obtained a Grade C is: 2
The number of students who have obtained a Grade D is: 0
The number of students who have obtained a Grade F is: 1

The student with the highest percentage is James McKay with a percentage of 93%.
The student with the lowest percentage is Kym Munro with a percentage of 29%.
```

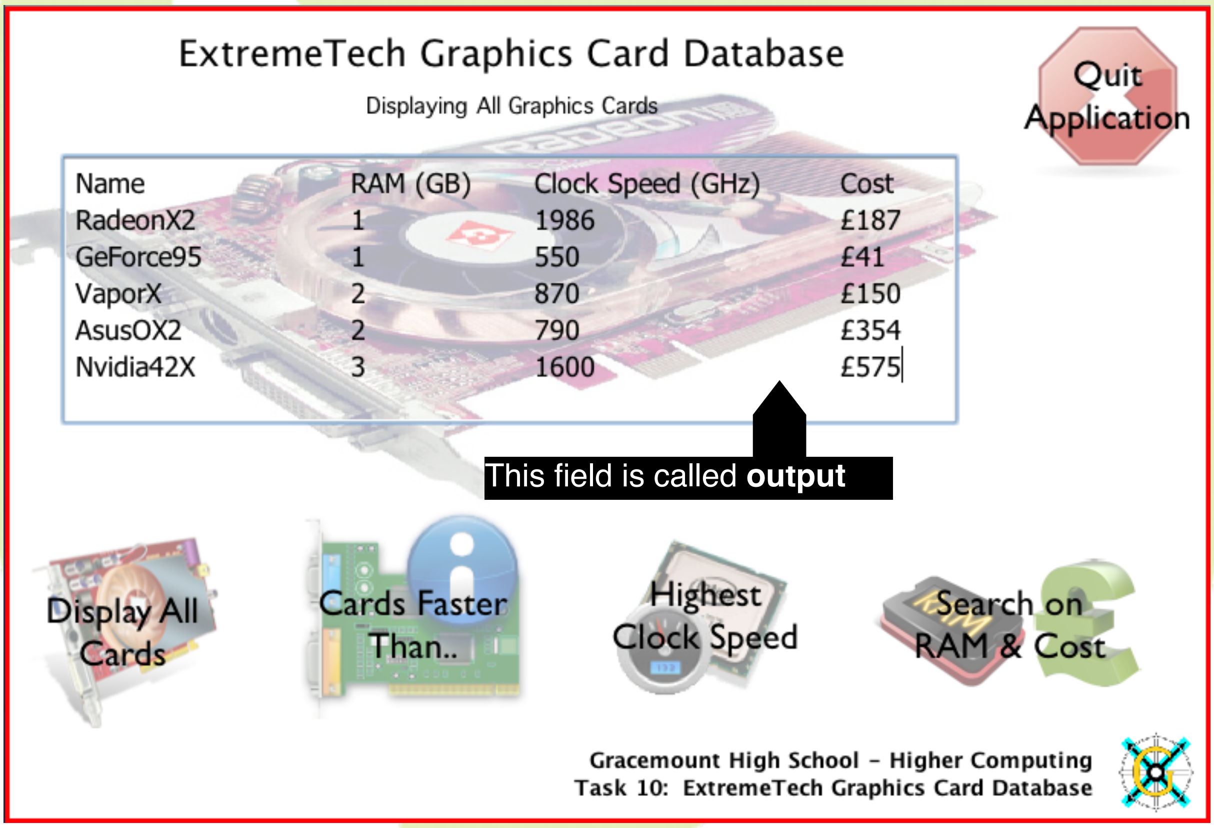
Task 10: ExtremeTech Graphics Card Database

Specification

A program is required by a company called ExtremeTech. The company specialise in the selling of high quality graphics cards.

ExtremeTech require the program in their retail outlets to allow:

- The customer to **display** a list of all **graphics cards** on the system.
- The customer to **search** for all graphics cards based on their **requirements** of how much the customer is willing to spend on a graphics card (**maximum cost**), and how much RAM it must have (**minimum RAM**).
- The program should **find** and display the **name** and **clock speed** of the graphics card with the **highest** clock speed.
- The program should also allow the user to **find** the **number** of graphics cards that are **higher** than the **threshold clock speed** that the user enters.


ExtremeTech Graphics Card Database

Displaying All Graphics Cards

```
Name        RAM (GB)   Clock Speed (GHz)   Cost
RadeonX2    1          1986                £187
GeForce95   1          550                 £41
VaporX      2          870                 £150
AsusOX2     2          790                 £354
Nvidia42X   3          1600                £575
```

This field is called **output**
Quit Application
Display All Cards
Cards Faster Than..
Highest Clock Speed
Search on RAM & Cost
Gracemount High School - Higher Computing
Task 10: ExtremeTech Graphics Card Database

Task 10: ExtremeTech Graphics Card Database

Design for “Display All Cards” Graphic

Stepwise Design *(the main steps of the program with data flow)*

1.	Setup	**In/Out**: arrayname, arrayram, arrayclockspeed, arraycost
		In/Out: username
2.	Display Data	**In**: arrayname, arrayram, arrayclockspeed, arraycost

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup
1.1     Setup arrayname, arrayram, arrayclockspeed, arraycost as global arrays
1.2     Setup username as global variable
1.4     Clear the “subheading” and “output” fields
1.5     Ask the user for their name
1.6     Put it into the variable username
1.7     Put “RadeonX2”, GeForce95”, “VaporX”, “AsusOX2”, “Nvidia42X” into array_name
1.8     Split arrayname using a comma
1.9     Put 1, 1, 2, 2, 3 into arrayram
1.10    Split the arrayram using a comma
1.11    Put 1986, 550, 870, 790, 1600 into arrayclockspeed
1.12    Split arrayclockspeed using a comma
1.13    Put 187, 41, 150, 354, 575 into arraycost
1.14    Split arraycost using a comma

2.    Display Data
2.1     Put “Displaying All Graphics Cards” into the field “subheading”
2.2     Put “Name” tab “RAM (GB)” tab “Clock Speed (GHz)” tab “Cost” into line 1 of field “output”
2.3     Repeat with loop 1 to 5
2.4       Put arrayname[loop], tab arrayram[loop], tab arrayclockspeed [loop], tab arraycost [loop] into
          line loop + 1 of field “output”
2.5     End repeat
```

Please **READ** the following before you begin the first script.

After carefully reading through the design above. You should begin to code the script for the first button called “Get Student Details”. Key in all of the code over the page **carefully**.

After completing **each** event, you should **test** that your program is **working correctly** by displaying a list of all graphics cards in the output field. The output should be the **same** as shown on the right.

Open the “**ExtremeTech Graphics Card Database**” stack. It can be found on Glow:

LiveCode Programming Tasks > 10_ExtremeTech Graphics Card Database.livecode

Task 10: ExtremeTech Graphics Card Database

Implementation: Script “Display All Cards”

Key the following code into the **Display All Cards** button. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Setup the global arrays and variable to be used in this event
global arrayname, arrayram, arrayclockspeed, arraycost, username

on mouseup
   set_up arrayname, arrayram, arrayclockspeed, arraycost, username
   display_data arrayname, arrayram, arrayclockspeed, arraycost
end mouseup

on set_up @arrayname, @arrayram, @arrayclockspeed, @arraycost, @username
   put empty into field "subheading"
   put empty into field "output"
   ask "Please enter your name"
   put it into username

   // Add the graphics card data to the arrays
   put "RadeonX2","GeForce95","VaporX","AsusOX2","Nvidia42X" into arrayname
   split arrayname by comma
   put 1,1,2,2,3 into arrayram
   split arrayram by comma
   put 1986,550,870,790,1600 into arrayclockspeed
   split arrayclockspeed by comma
   put 187,41,150,354,575 into arraycost
   split arraycost by comma
end set_up

on display_data arrayname, arrayram, arrayclockspeed, arraycost
   put "Displaying All Graphics Cards" into field "subheading"
   put "Name" & tab & "RAM (GB)" & tab & "Clock Speed (GHz)" & tab & "Cost" into line 1 of field
   "output"  // On the same line
   repeat with loop = 1 to 5
      put arrayname[loop] & tab & arrayram[loop] & tab & arrayclockspeed[loop] & tab & "£" &
      arraycost[loop] into line loop+1 of field "output"  // On the same line
   end repeat
    // Enable the following buttons when "Display All Cards" is selected
    enable image buttonFasterThan
    enable image buttonHighestClockSpeed
    enable image buttonRAMandCost
    enable image buttonQuitApplication
end display_data
```

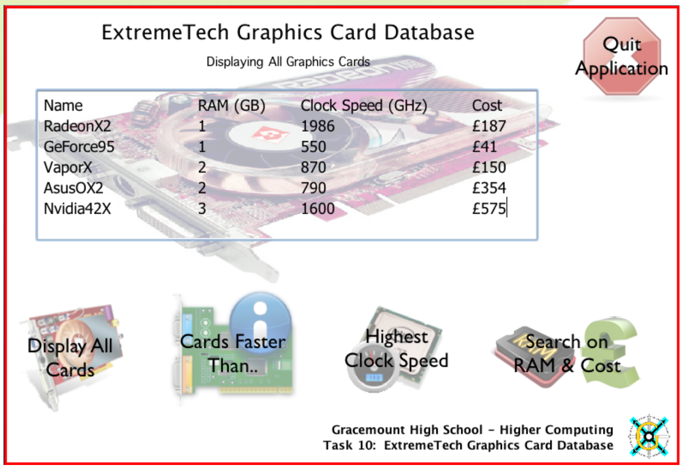
Testing

Test that your program correctly produces a list of the five graphics cards when you run your program and select the “Display All Cards” button.

Your output should look the same as that shown on the right.

If it works correctly, move onto the next part of the program.

If your program is not working correctly, **correct** your **errors** and **retest**.


ExtremeTech Graphics Card Database

Displaying All Graphics Cards

```
Name        RAM (GB)   Clock Speed (GHz)   Cost
RadeonX2    1          1986                £187
GeForce95   1          550                 £41
VaporX      2          870                 £150
AsusOX2     2          790                 £354
Nvidia42X   3          1600                £575
```

Task 10: ExtremeTech Graphics Card Database

Design for “Cards Faster Than..” Graphic

Stepwise Design *(the main steps of the program with data flow)*

```
1.   Setup
2.   Cards Faster Than      In: arrayclockspeed
```

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.     Setup
1.1      Pass in arrayclockspeed as the global array to be used in this event

1.3      Clear the “subheading” and “output” fields
1.6      Put the heading “Find Graphics Cards Faster Than..” into field “subheading”

2.     Cards Faster Than
2.1      Setup the local variables of min_clock_speed and counter to be used in this subroutine
2.2      Put 0 into min_clock_speed
2.3      Put 0 into counter
2.4      Ask the user to enter the minimum clock speed
2.5      Put the minimum clock speed into the variable minclockspeed
2.6      Repeat with loop 1 to 5
2.7        If arrayclockspeed [loop] is greater than minclockspeed then
2.8        Add 1 to the variable counter
2.9        End If
2.10     End Repeat

2.11     Put the message “The number of graphics cards faster than ” followed by the variable
         minclockspeed “MHz is ” followed by the variable counter into line 1 of the field “output”
```

Please **READ** the following before you begin the second script.

After carefully reading through the design above. You should begin to code the script for the first button called “Cards Faster Than..”. Key in all of the code over the page **carefully**.

After completing **each** event, you should **test** that your program is **working correctly**.

The program should take in a **minimum clock speed** from the user and using the **counting occurrences** algorithm, display the **number** of graphics cards that are faster than the minimum clock speed.

This should be placed into line 1 of the output field.

Task 10: ExtremeTech Graphics Card Database

Implementation: Script “Cards Faster Than..”

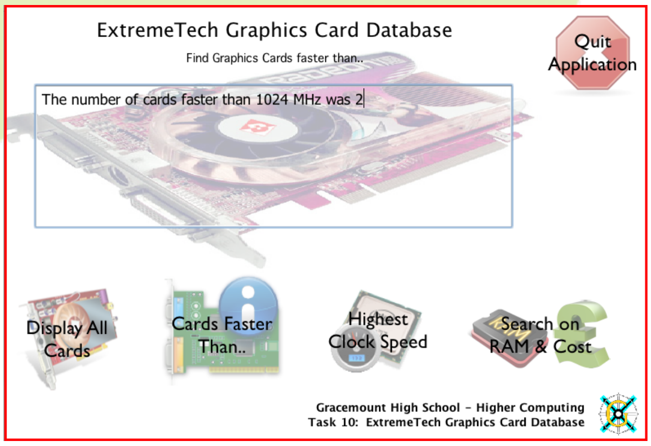
Key the following code into the **Cards Faster Than..** button. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
//  Pass in the global arrays setup earlier
global arrayclockspeed

on mouseUp
   set_up
   cards_faster_than arrayclockspeed
end mouseUp

on set_up
   //  Clear text from the fields
   put empty into field "subheading"
   put empty into field "output"

   //  Display the heading
   put "Find Graphics Cards faster than.." into field "subheading"
end set_up

on cards_faster_than arrayclockspeed
    // Set up the local variables
   local minclockspeed, counter

  put 0 into minclockspeed
   put 0 into counter

   //  Ask for the minimum clock speed
   ask "What is the minimum speed?"
   put it into minclockspeed

   // Display graphics cards matching search criteria
   repeat with loop = 1 to 5
      if arrayclockspeed[Loop] > minclockspeed then
         add 1 to counter
      end if
   end repeat
   put "The number of cards faster than "&  minclockspeed & " MHz was "&counter into line 1 of field
   "output"  // On the same line
end cards_faster_than
```

Testing

Test that your program correctly produces displays the number of graphics cards faster than the minimum clock speed entered by the user.

The output on the right shows the number of graphics cards faster than **1024MHz**. **2** is the correct answer as both the **RadeonX2** and the **Nvidia42X** have a **faster clock speed** of **1024MHz**.


ExtremeTech Graphics Card Database
Find Graphics Cards faster than..
Quit Application
Display All Cards
Cards Faster Than..
Highest Clock Speed
Search on RAM & Cost
Gracemount High School – Higher Computing
Task 10: ExtremeTech Graphics Card Database

The number of cards faster than 1024 MHz was 2

You should do **similar testing** and **check** with the **list** of **cards** if the **number** produced is **correct**. If it is, move onto the next part of this program.

If your program is not working correctly, **correct** your **errors** and **retest**.

Task 10: ExtremeTech Graphics Card Database

Design for “Highest Clock Speed” Graphic

Stepwise Design *(the main steps of the program with data flow)*

```
1.    Setup
2.    Highest Clock Speed          In:  arrayname, arrayclockspeed
```

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup
1.1     Pass in arrayname, arrayclockspeed as the global arrays to be used in this event

1.2     Clear the “subheading” and “output” fields
1.3     Put the heading “Find the Graphics Card with the Highest Clock Speed” into field “subheading”

2.    Highest Clock Speed
2.1     Setup the local variables of maxclockspeed and position to be used in this subroutine
2.2     Put 0 into maxclockspeed
2.3     Put 0 into position

2.4     Repeat with loop 1 to 5
2.5        If arrayclockspeed [loop] is greater than maxclockspeed then
2.6        Put arrayclockspeed[loop] into maxclockspeed
           Put loop into position
2.7        End If
2.8     End Repeat

2.9     Put the message “The card with the highest clock speed is ” followed by arrayname[position] into
        line 1 of field “output”
```

Please **READ** the following before you begin the second script.

After carefully reading through the design above. You should begin to code the script for the first button called “Highest Clock Speed”. Key in all of the code over the page **carefully**.

After completing **each** event, you should **test** that your program is **working correctly**.

The program should find the graphics card with the **highest clock speed** using the **find maximum** algorithm and then display the **name** of graphics card in line 1 of the output field.

Task 10: ExtremeTech Graphics Card Database

Implementation: Script “Highest Clock Speed”

Key the following code into the **Highest Clock Speed** button. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Allow access to global arrays setup earlier
global arrayname, arrayclockspeed

on mouseUp
   set_up
   highest_clock_speed arrayname, arrayclockspeed
end mouseUp

on set_up
   // Clear text from the fields
   put empty into field "subheading"
   put empty into field "output"

   // Display the heading
   put "Find the Graphics Card with the Highest Clock Speed" into field "subheading"
end set_up

on highest_clock_speed arrayname, arrayclockspeed
   // Set up the local variables
   local maxclockspeed, position

   put 0 into maxclockspeed
   put 0 into position

   // Display graphics cards matching search criteria
   repeat with loop = 1 to 5
      if arrayclockspeed[Loop] > maxclockspeed then
         put arrayclockspeed[Loop] into maxclockspeed
         put loop into position
      end if
   end repeat
   put "The card with the highest clock speed is "& arrayname[position] into line 1 of field
"output"  // On the same line
end highest_clock_speed
```

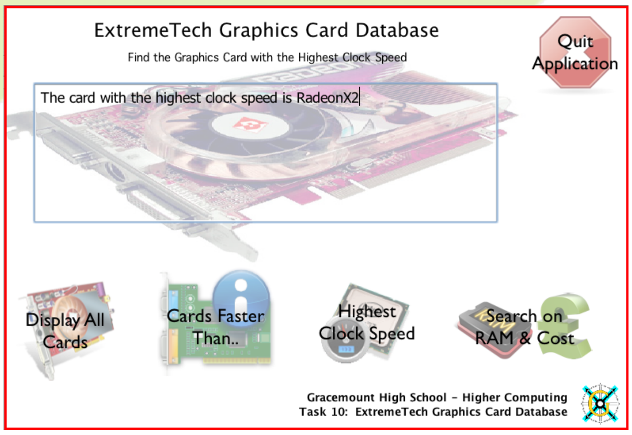
Testing

Test that your program correctly displays the **name** of the **graphics card** with the **fastest clock speed**.

Your output should be the **same** as shown on the right. This shows that the **RadeonX2** is the graphics card with the **fastest clock speed**.

If the correct output is produced, move onto the next part of this program.

If your program is not working correctly, **correct** your **errors** and **retest**.


ExtremeTech Graphics Card Database

Find the Graphics Card with the Highest Clock Speed

The card with the highest clock speed is RadeonX2

Gracemount High School – Higher Computing
Task 10: ExtremeTech Graphics Card Database

Task 10: ExtremeTech Graphics Card Database

Design for “Search on RAM and Cost” Graphic

Stepwise Design *(the main steps of the program with data flow)*

```
1.  Setup
2.  Search Cards  In: arrayname, arrayram, arrayclockspeed, arraycost
```

Stepwise Refinement *(the main steps further refined into smaller steps)*

```
1.    Setup
1.1     Pass in arrayname, arrayram, arrayclockspeed and arraycost as the global arrays to be used in
        this event

1.2     Clear the “subheading” and “output” fields
1.3     Put the heading “Find the Graphics Card that match RAM and Cost” into field “subheading”

2.    Search Cards
2.1     Setup the local variables of minram and maxcost to be used in this subroutine
2.2     Ask the user to enter the minimum amount of RAM required
2.3     Put it into the variable minram
2.4     Ask the user to enter the maximum amount that they are willing to spend on a graphics card
2.5     Put it into the variable maxcost
2.6     Put “Name” tab “RAM (GB)” tab “Clock Speed (GHz)” tab “Cost” into line 1 of field “output”
2.7     Repeat with loop 1 to 5
2.8        If arrayram[loop] >= minram and arraycost[loop] is <= maxcost then
2.9           Put arrayname[loop], tab arrayram[loop], tab arrayclockspeed [loop], tab arraycost [loop]
              into line loop + 1 of field “output”
2.10       End If
2.11    End Repeat
```

Please **READ** the following before you begin the second script.

After carefully reading through the design above. You should begin to code the script for the first button called “Search on RAM and Cost”. Key in all of the code over the page **carefully**.

After completing **each** event, you should **test** that your program is **working correctly**.

The program should find the graphics cards based on the minimum RAM and maximum cost that the user enters using the **linear search** algorithm. It should display a list of graphics cards which match the search in the output field.

Task 10: ExtremeTech Graphics Card Database

Implementation: Script “Search on RAM and Cost”

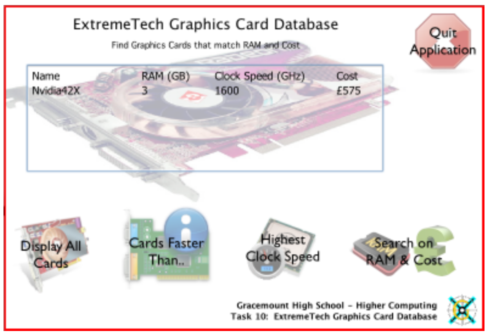
Key the following code into the **Highest Clock Speed** button. You **don’t** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Allow access to global arrays setup earlier
global arrayname, arrayram, arrayclockspeed, arraycost

on mouseUp
  set_up
  search_cards arrayname, arrayram, arrayclockspeed, arraycost
end mouseUp

on set_up
  // Clear text from the fields
  put empty into field "subheading"
  put empty into field "output"
  // Display the heading
  put "Find Graphics Cards that match RAM and Cost" into field "subheading"
end set_up

on search_cards arrayname, arrayram, arrayclockspeed, arraycost
  // Set up the local variables
  local minram
  local maxcost
  local found
  // Ask the user for the minimum cost and maximum amount of RAM required
  ask "Please enter the minimum amount of RAM you wish your graphics card to have:"
  put it into minram
  ask "Please enter the maximum amount you are willing to spend on a new graphics card:"
  put it into maxcost

  // Display graphics cards matching search criteria
  put "Name" & tab & "RAM (GB)" & tab & "Clock Speed (GHz)" & tab & "Cost" into line 1 of field "output"
  put false into found
  repeat with loop = 1 to 5
    if arrayram[Loop] >= minram AND arraycost[loop] <= maxcost then
      put arrayname[loop] & tab & arrayram[loop] & tab & arrayclockspeed[loop] & tab & "£" & arraycost
      [loop] into line loop+3 of field "output"  // On the same line
      put true into found
    end if
  end repeat
  if found = false then put "****** No Cards which match your search have been found ******" into line
  position of field "output"  // On the same line
end search_cards
```

Testing

Test that your program correctly displays a **list** of graphics cards based on the users **search** of **RAM** and **Cost**.

The output shown on the right assumes a **minimum RAM** of **3GB** and a **maximum cost** of **£800**.



ExtremeTech Graphics Card Database
Find Graphics Cards that match RAM and Cost
Quit Application
Display All Cards
Cards Faster Than..
Highest Clock Speed
Search on RAM & Cost
Gracemount High School – Higher Computing
Task 10: ExtremeTech Graphics Card Database

```
Name        RAM (GB)   Clock Speed (GHz)   Cost
Nvidia42X   3          1600                £575
```

If the correct output is produced, move onto the next part of this program.

If your program is not working correctly, **correct** your **errors** and **retest**.