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.

National 5 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** software project and are summarised below.

Analysis A

A statement about **what** your program is going to do. You will also include a description of the **main steps** of the problem.

Design Dance

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

For the purpose of Intermediate 2 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

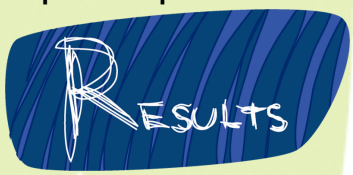
The implementation stage involves keying in the program code using the built in **text editor** within the programming environment. We will use **LiveCode** to create our programs.

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

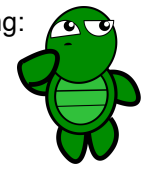
Documentation is usually produced in the form of a **user guide** and a **technical guide**. The user guide shows the user how to **use** the **functions** and **features** of the **software** whereas the **technical guide** gives the user information on how to **install** the **software** as well as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



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

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 a **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**. The design language used for the refinements is called **HAGGIS** and is a requirement of the SQA.

Here is the program pseudocode to calculate the volume of a room using the variables **Room_Length**, **Room_Breadth**, **Room_Height** and **Room_Volume**. Study the **pseudocode** very closely to understand what is going on:

HAGGIS Design Language

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

1. Setup Variables
2. Get Room Measurements
3. Calculate Room Volume
4. Display Room Volume

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

1. Setup Variables
1.1 **SET** Room_Length **TO** Real 0.00
1.2 **SET** Room_Breadth **TO** Real 0.00
1.3 **SET** Room_Height **TO** Real 0.00
1.4 **SET** Room_Volume **TO** Real 0.00

2. Get Room Measurements
2.1 **RECEIVE** Room_Length **FROM (Real) KEYBOARD**
2.2 **RECEIVE** Room_Breadth **FROM (Real) KEYBOARD**
2.3 **RECEIVE** Room_Height **FROM (Real) KEYBOARD**

3. Calculate Room Volume
3.1 **SET** Room_Volume **TO** Room_Length * Room_Breadth * Room_Height

4. Display Room Volume
4.1 **SEND** Room_Volume **TO DISPLAY**

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?

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

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

To put it simply, a variable is like a "box" into which data can be placed whilst a program is **running**. We give variables 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 *Hello* 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.
- **Boolean** (**two state** values), e.g. Yes/No, True/False, 1/0, etc.

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

Declaring Variables in LiveCode

```
Put 0.00 into Room_Length
// Setup a Real variable called Room_Length and put 0.00 into this variable

Put "" into Player_Name
// Setup a String variable called Player_Name and put "" into this variable

Put 0 into Number_Correct
// Setup an Integer variable called Number_Correct and put 0 into this variable

Put False into Found
// Setup a Boolean variable called Found and put False into this variable

Put True into Found
// Setup a Boolean variable called Found and put True into this variable
```

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
```

← *This is the variable*

In order for the program to know which data is a **variable** and which is **text** to be **printed** in a **put statement**, the ampersand **&** is used.

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 be using **both types** of **variable**.

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.

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 very portable which means that it can run on a **variety** of operating system **platforms**. This includes a **PC** running Windows XP, Vista, Windows 7 and 8 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 (mac) or **Program Files** (Windows PC).

IMPORTANT

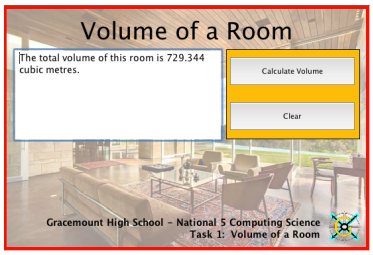
You will need to **copy** the **LiveCode Programming Tasks** from the **National 5 LiveCode Programming** area of Glow to **your own programming** folder within your user folder. This folder contains the National 5 LiveCode stacks for all 10 tasks that you will do during this programming unit.

N5 > Software Design & Development > N5 LiveCode Programming

LiveCode
Programming Tasks

Task 1: Volume of a Room

Specification



A simple program 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

HAGGIS Design Language

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

1. **Setup Variables**
2. **Get Room Measurements**
3. **Calculate Room Volume**
4. **Display Room Volume**

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

1. Setup Variables
1.1 **SET** Room_Length **TO** Real 0.00
1.2 **SET** Room_Breadth **TO** Real 0.00
1.3 **SET** Room_Height **TO** Real 0.00
1.4 **SET** Room_Volume **TO** Real 0.00

2. Get Room Measurements
2.1 **RECEIVE** Room_Length **FROM (Real) KEYBOARD**
2.2 **RECEIVE** Room_Breadth **FROM (Real) KEYBOARD**
2.3 **RECEIVE** Room_Height **FROM (Real) KEYBOARD**

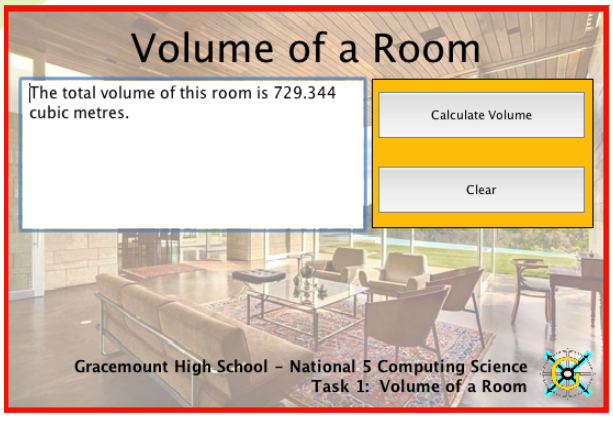
3. Calculate Room Volume
3.1 **SET** Room_Volume **TO** Room_Length * Room_Breadth * Room_Height

4. Display Room Volume
4.1 **SEND** Room_Volume **TO DISPLAY**

Implementation

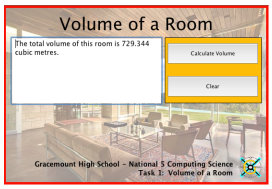
After reading through the above design carefully, you are now ready to begin producing your program code.

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 in:

N5 LiveCode Programming > 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
  Setup_Variables
  Get_Room_Measurements
  Calculate_Room_Volume
  Display_Room_Volume
End MouseUp

On Setup_Variables
  // Setup variables to 0.00 (real data types)
  Put 0.00 into Room_Length
  Put 0.00 into Room_Breadth
  Put 0.00 into Room_Height
  Put 0.00 into Room_Volume
End Setup_Variables

On Get_Room_Measurements
  // 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
  // Calculate the volume of the room
  Put (Room_Length * Room_Breadth * Room_Height) into Room_Volume
End Calculate_Room_Volume

On Display_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
```

Make sure you have selected the "**Calculate Volume**" button and are in **Edit Mode**.

Run Mode **Edit Mode**

Select the "**Code**" button at the top left of the toolbar.

You will now enter the lines of program code on the left page **very carefully**.

Testing

Once you have got your program running, remember to test that it works correctly by producing the correct Output.

LiveCode Commands and Loops

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:

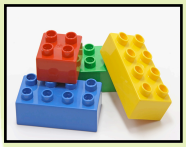
```
Ask "Please enter the length of the room in metres"
```

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
```



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
  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
```

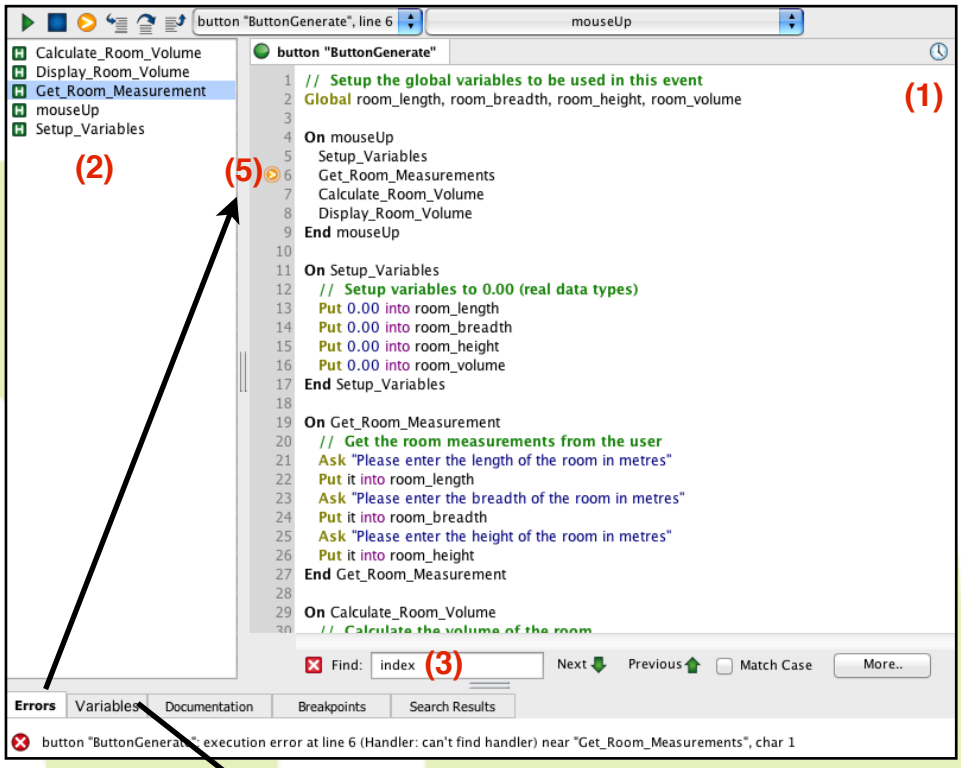
The LiveCode Text Editor & Error Checking

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

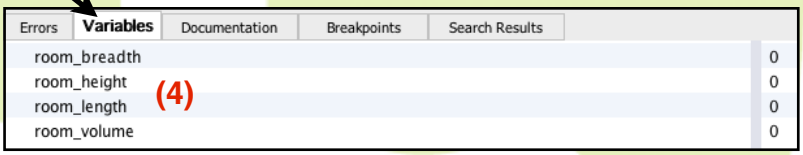
The LiveCode text editor has a wide range of editing features to help make programming easier **(1)**.

The LiveCode text editor gives you a list of all the events you have created within the program **(2)**.

The LiveCode text editor also has the ability to find any term in a program **(3)**.

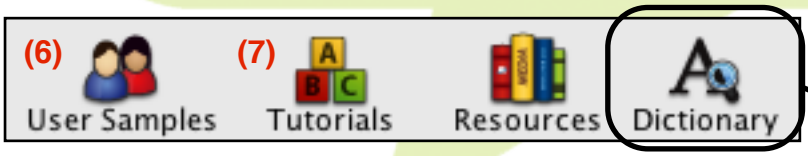


The LiveCode text editor will also give you a complete list of all variables and arrays used in the program **(4)**.

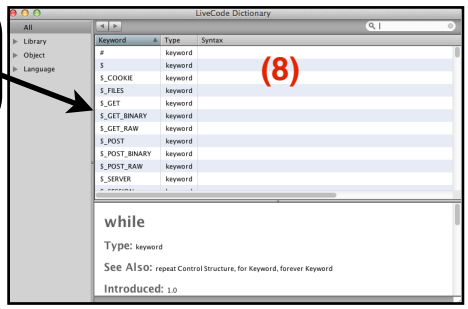


During implementation of a program, LiveCode uses an **interpreter** translator program to help check for errors (see your Software Design and Development notes for a description of this type of translator program).

In the above program, the user is told of an error on line 6 “Get_Room_Measurements”. The user needs to look at this line then scan down the program to see that they have in fact called the event on lines 19 and 27 “Get_Room_Measurement” **(5)**.



LiveCode comes with uploaded User Sample **(6)** programs and Online Tutorials **(7)** to help them with their programming.



The LiveCode toolbar comes with lots of ways of accessing LiveCode available commands and resources. The Dictionary, can be called upon to gain access to all of the LiveCode commands **(8)**.

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 calculate and display the total takings, based on the prices of CD's (£7.99), DVD's (£10.99) and Blu-Ray Disks (£14.99).

Design: Pseudocode for "Go" Button

HAGGIS Design Language

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

1. Setup Variables
2. Get Number Of Items Sold
3. Calculate Total Takings
4. Display Total Takings

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

1. Setup Variables
1.1 **SET** CDs_Sold **TO** Integer 0
1.2 **SET** DVDs_Sold **TO** Integer 0
1.3 **SET** BluRays_Sold **TO** Integer 0
1.4 **SET** Total_Takings **TO** Real 0.00

2. Get Number Of Items Sold
2.1 **RECEIVE** CDs_Sold **FROM (Integer) KEYBOARD**
2.2 **RECEIVE** DVDs_Sold **FROM (Integer) KEYBOARD**
2.3 **RECEIVE** BluRays_Sold **FROM (Integer) KEYBOARD**

3. Calculate Total Takings
3.1 **SET** Total_Takings **TO** CDs_Sold * 7.99 + DVDs_Sold * 10.99 + BluRays_Sold * 14.99

4. Display Total Takings
4.1 **SEND** CDs_Sold **TO DISPLAY**
4.2 **SEND** DVDs_Sold **TO DISPLAY**
4.3 **SEND** BluRays_Sold **TO DISPLAY**
4.4 **SEND** Total_Takings **TO DISPLAY**

Task 2: Music Shop Takings

Implementation

Open the "**Music Shop Takings**" stack. It can be found in:

N5 LiveCode Programming > 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**.

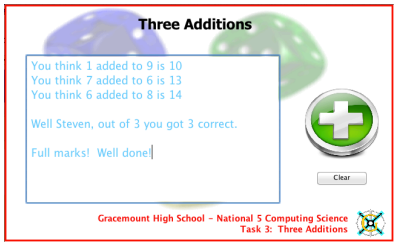
```
// Setup the global variables to be used in this event
Global CDs_Sold, DVDs_Sold, BluRays_Sold, Total_Takings

On MouseUp
  Setup_Variables
  Get_Number_Of_Items_Sold
  Calculate_Total_Takings
  Display_Total_Takings
End MouseUp

On Setup_Variables
  // Initialise the variables
  Put 0 into CDs_Sold
  Put 0 into DVDs_Sold
  Put 0 into BluRays_Sold
  Put 0.00 into Total_Takings
End Setup_Variables

On Get_Number_Of_Items_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
  // Calculate the total cost by taking the amount of items sold by the user and multiplying it by
  // the of each product
  Put CDs_Sold * 7.99 + DVDs_sold * 10.99 + BluRays_Sold * 14.99 into Total_Takings
End Calculate_Total_Takings

On Display_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"

  // Display the total cost
  Put "Today you have made a total of £" & Total_Takings into field "Output2"
End Display_Total_Takings
```

Testing

Once you have got your program running, remember to test that it works correctly by producing the correct Output.

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 **1** 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 an Output field.

Design: Pseudocode for "Plus" graphic

HAGGIS Design Language

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

1. Setup Variables
2. Get The Users Name
3. Three Additions

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

```
1.    Setup Variables
1.1     SET Name_Of_Person TO String ""
1.2     SET First_Number TO Integer 0
1.3     SET Second_Number TO Integer 0
1.4     SET The_Answer TO Integer 0

2.    Get The Users Name
2.1     SET Key_Pressed TO local variable
2.2     REPEAT until Key_Pressed = "Y" OR Key_Pressed ="y"
2.3       RECEIVE Name_Of_Person FROM (string) KEYBOARD
2.4       RECEIVE "Y" or "N" FROM (String) KEYBOARD
2.5     END REPEAT

3.    Three Additions
3.1     SET Number_Correct TO local variable
3.2     SET Number_Correct TO integer 0
3.3     REPEAT with loop = 1 to 3
3.4       SET First_Number TO Random (0-9)
3.5       SET Second_Number TO Random (0-9)
3.6       RECEIVE the_answer FROM (integer) KEYBOARD
3.7       IF cancel button is pressed THEN exit
3.8       SEND The_Answer TO DISPLAY
3.9       IF The_Answer =  First_Number + Second_Number THEN
3.10        SEND "Correct Answer, Well Done!" TO DISPLAY
3.11        SET Number_Correct TO Number_Correct + 1
3.12      ELSE
3.13        SEND "Wrong Answer." TO DISPLAY
3.14      END IF
3.15    END REPEAT

3.16    SEND How many questions out of 3 the user got correct TO DISPLAY
3.17    IF Number_Correct is = 0 THEN SEND "Very disappointing" TO DISPLAY
3.18    IF Number_Correct is = 1 THEN SEND "Disappointing" TO DISPLAY
3.19    IF Number_Correct is = 2 THEN SEND "Good work" TO DISPLAY
3.20    IF Number_Correct is = 3 THEN SEND "Full Marks, Well done!" TO DISPLAY
```

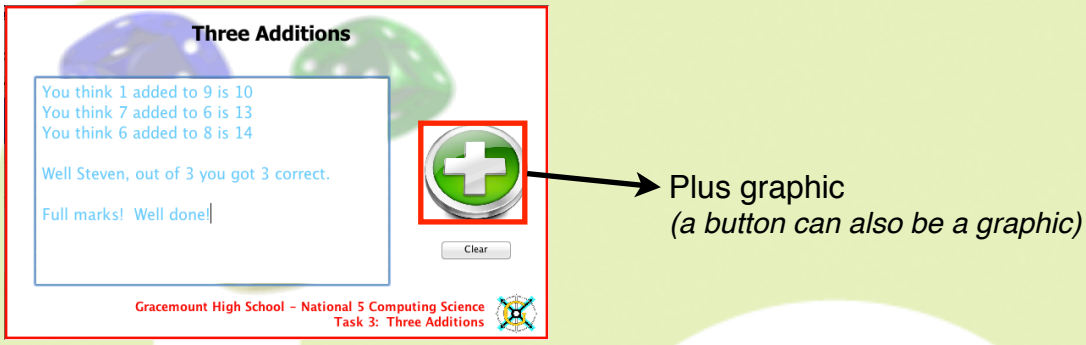
Task 3: Three Additions

Implementation

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

N5 LiveCode Programming > 3_Three Additions.livecode

Enter 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.




```
// Setup the global variables to be used in this event
Global Name_Of_Person, First_Number, Second_Number, The_Answer

On MouseUp
  Setup_Variables
  Get_Users_Name
  Three_Additions
End MouseUp

On Setup_Variables
  // Setup all the variables to null or 0
  Put "" into Name_Of_Person
  Put 0 into First_Number
  Put 0 into Second_Number
  Put 0 into The_Answer
End Setup_Variables

On Get_Users_Name
  // 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. Y or y indicates they are happy and loop is exited.
  REPEAT until Key_Pressed = "Y" or Key_Pressed = "y"
    Ask "Please enter your name"
    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
  //  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 The_Answer

    Put "So you think " & First_Number & " added to " & Second_Number & " is "
    & The_Answer into line loop of field "Output"  //  On the same line

    IF The_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 "Very Disappointing" into line 7 of field "Output"
  IF Number_Correct = 1 THEN Put "Disappointing" into line 7 of field "Output"
  IF Number_Correct = 2 THEN Put "Good work!" into line 7 of field "Output"
  IF Number_Correct = 3 THEN Put "Full Marks, well done!" into line 7 of field "Output"

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**.

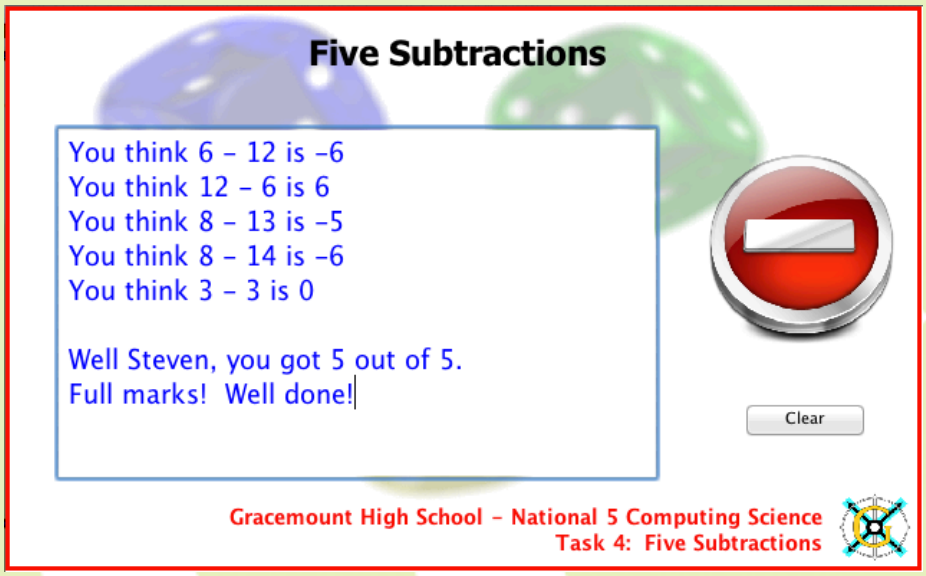
Task 4: Five Subtractions

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:



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**.
- 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 in:

N5 LiveCode Programming > 4_Five Subtractions.livecode

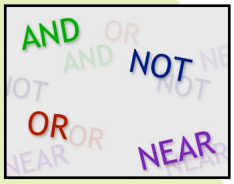
Good Luck!

Logical Operators

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

Logical Operators like **AND**, **OR** and **NOT** are very useful for **decision making**. They are usually used within an **IF statement**.

Below is an example of how the **AND** and **OR** operators can be used. In the example below, the user is asked three questions on capital cities. Using the AND and OR operators, a decision can be made as to whether the user is getting the questions correct. Read the following program code carefully.




```
// Setup the global variables to be used in this event
global Answer1, Answer2, Answer3

On MouseUp
  Setup_Variables
  Ask_Questions
  Decide
End MouseUp

on Setup_Variables
  // Setup the string variables to be used in this program
   Put "" into Answer1
   Put "" into Answer2
   Put "" into Answer3
end Setup_Variables

on Ask_Questions
  // Ask the user three simple Capital City questions
  Ask "What is the Capital City of France?"
  Put it into Answer1
  Ask "What is the Capital City of Spain?"
  Put it into Answer2
  Ask "What is the Capital City of Germany?"
  Put it into Answer3
end Ask_Questions

on Decide
  // IF the answer to France is equal (=) to Paris AND the answer to Spain is equal to Madrid AND
  // the answer to Germany is equal to Berlin then the user has got all of the answers correct.
  // Display a suitable message in the field called output.

  IF Answer1 = "Paris" AND Answer2 = "Madrid" AND Answer3 = "Berlin" THEN
     Put "Well done, you got all the Capital Cities correct!!" into field "output"
  END IF

  // IF the answer to France is NOT equal (<>) to Paris OR the answer to Spain is NOT equal to
  // Madrid OR the answer to Germany is NOT equal to Berlin then the user has either got one or
  // two of the answers incorrect. Display a suitable message in the field called output.

  IF Answer1 <> "Paris" OR Answer2 <> "Madrid" OR Answer3 <> "Berlin" THEN
     Put "Good effort, you didn't get all the Capital Cities correct though." into field "output"
  END IF

  // IF the answer to France is NOT equal (<>) to Paris AND the answer to Spain is NOT equal to
  // Madrid AND the answer to Germany is NOT equal to Berlin then the user has
  // got all of the answers incorrect. Display a suitable message in the field called output.

  IF Answer1 <> "Paris" AND Answer2 <> "Madrid" AND Answer3 <> "Berlin" THEN
     Put "Poor show! You got none of the Capital Cities correct" into field "output"
  END IF

End decide
```

Task 5: Iain's iPod Takings

Specification

Iain would like a program created which will calculate the **total amount** of sales of iPods in his shop on a daily basis and give the employee that is working on that day a special bonus if they sell over a certain amount of iPods.

The three types of Apple iPods he sells are shown below along with their price.

iPod Touch
£220.00

iPod Classic
£150.00

iPod Nano
£90.00

The program should take in the amount of **iPod Touch's**, **iPod Classic's** and **iPod Nano's** sold over the course of the day.

The **total cost** of all iPods sold should be calculated and displayed.

In order for the employee to get a bonus on the sales they make from the iPods, they must sell at least **12 iPod Touch's and either 8 iPod Classic's or 10 iPod Nano's**.

If they meet the above criteria, then a suitable message should be displayed telling them that they **should get a bonus**.

Else, they should be told that sales were not high enough for that day and that they **should not get a bonus**.

Task 5: Iain's iPod Takings

Design

The design of the program is shown below:

HAGGIS Design Language

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

1. **Setup Variables**
2. **Get Number Sold**
3. **Calculate Total Cost**
4. **Display and Decide**

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

1. Setup Variables
1.1 **SET** Touch **TO** Integer 0
1.2 **SET** Classic **TO** Integer 0
1.3 **SET** Nano **TO** Integer 0
1.4 **SET** Total **TO** Integer 0

2. Get Number Sold
2.1 **RECEIVE** Touch **FROM (Integer) KEYBOARD**
2.2 **RECEIVE** Classic **FROM (Integer) KEYBOARD**
2.3 **RECEIVE** Nano **FROM (Integer) KEYBOARD**

3. Calculate Total Cost
3.1 **SET** Total **TO** Touch * 220 + Classic * 150 + Nano * 90

4. Display and Decide
4.1 **SEND** Touch **TO DISPLAY**
4.2 **SEND** Classic **TO DISPLAY**
4.3 **SEND** Nano **TO DISPLAY**
4.4 **SET** Number Format **TO** "00.00"
4.5 **SEND** "£" followed by the contents of total **TO DISPLAY**
4.6 **IF** Touch >= 12 **AND** (Classic >= 8 **OR** Nano >=10) **THEN**
4.7 **SEND** You get a bonus on today's sales **TO DISPLAY**
4.8 **ELSE**
4.9 **SEND** Not enough iPods of each type have been sold to entitle you to a bonus. **TO DISPLAY**
4.10 **END IF**

Implementation

You are now going to use the design above to help code the solution for this program. It will also help to use the code from previous programs if you get stuck.

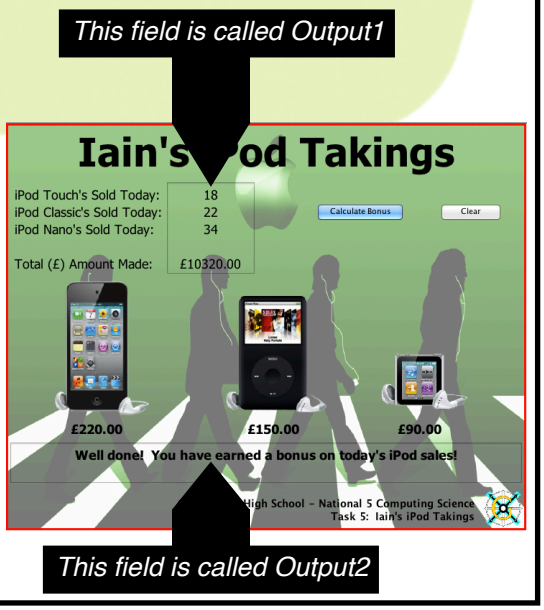
Open the LiveCode stack called "**Iain's iPod Takings**".

It can be found in:

N5 LiveCode Programming > 5_Iain's iPod Takings.livecode

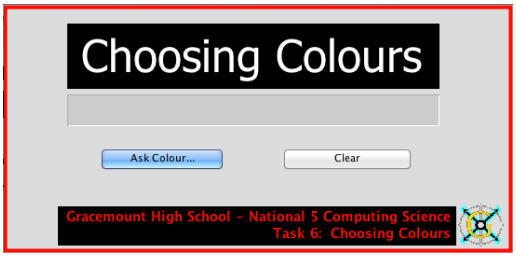
Assign your code to the **Calculate Bonus** button and **test** your program works correctly by calculating the total amount made and the correct decision. You can compare your Output to the screenshot shown on the right.

Good luck!



Task 6: 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

HAGGIS Design Language

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

1. Setup Variable
2. Choose Colour

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

1. Setup Variable
1.1 **SET** TheColour **TO** String ""

2. Choose Colour
2.1 **RECEIVE** TheColour **FROM (string) KEYBOARD**

2.2 **CASE** TheColour **OF**
2.3 **WHEN** "Red"
2.4 **SET** background and font colour **TO** red
2.5 **SEND** a message saying blood is red **TO DISPLAY**

2.6 **WHEN** "Blue"
2.7 **SET** background and font colour **TO** blue
2.8 **SEND** a message saying blood is blue **TO DISPLAY**

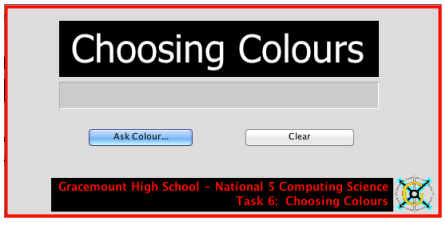
2.9 **WHEN** "Green"
2.10 **SET** background and font colour **TO** green
2.11 **SEND** a message saying blood is green **TO DISPLAY**

2.12 **WHEN** "Black"
2.13 **SET** background and font colour **TO** black
2.14 **SEND** a message saying blood is black **TO DISPLAY**

2.15 **WHEN** "Yellow"
2.16 **SET** background and font colour **TO** yellow
2.17 **SEND** a message saying blood is yellow **TO DISPLAY**
2.18 **END CASE**

2.19 **IF** the Output field is empty **THEN**
2.20 **SET** the background colour **TO** grey **AND** text colour **TO** black
2.21 **SEND** a message saying that there is no message for that colour **TO DISPLAY**
2.22 **END IF**

Implementation



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

N5 LiveCode Programming > 6_Choosing Colours.livecode

Task 6: Choosing Colours

Implementation

Add the following script to the **Ask Colour** button and **test** that your program produces the correct results for **each colour**.

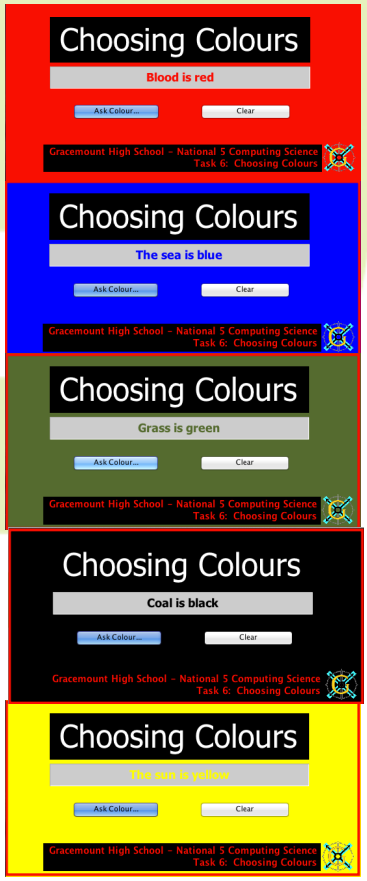
```
Global TheColour

On MouseUp
  Setup_Variable
  Choose_Colour
End MouseUp

On Setup_Variable
  //  Setup the variable to be used in this subroutine
  Put “” into TheColour
End Setup_Variable

On Choose_Colour
  //  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

  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
      Break
    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

  //  Set to default if no colour or an invalid colour is entered
  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
```

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	

Use the website **www.tayloredmktg.com/rgb/** to get the RGB colour code for the colours you have chosen.

After you have finished, add another **three colours** of your choice along with your own message for those colours.



Arrays (IMPORTANT)

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 **max_students** 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, max_students

On MouseUp
  // Number_of_Students will be set to five so five
  // names will be entered and stored in the array
  Put 5 into number_of_students
  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 arrayName
  REPEAT with loop = 1 to number_of_students
    // 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 number_of_students
    // 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
```

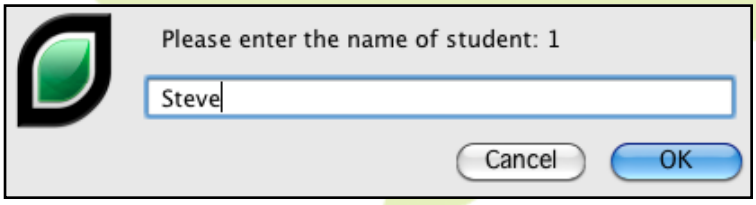
The program knows that **arrayName** is an **array** because of the **[loop]** straight after it.

[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 **max_students** 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)*:

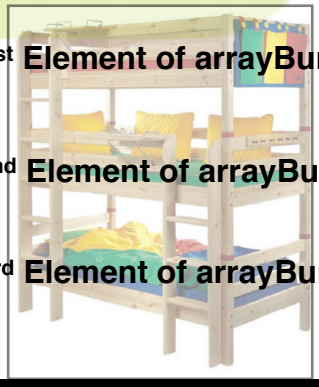
REPEAT with loop = 1 to **max_students**
arrayName[loop] - "*Steve*" - put into **1st** element of array
arrayName[loop] - "*Dave*" - put into **2nd** element of array
arrayName[loop] - "*Mike*" - put into **3rd** element of array
arrayName[loop] - "*Liam*" - put into **4th** element of array
arrayName[loop] - "*Allan*" - put into **5th** element of array
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**.



After the **five** names have been entered, the following Output will be produced using **arrayName**:

Steve
Dave
Mike
Liam
Allan



1st Element of arrayBunk

2nd Element of arrayBunk

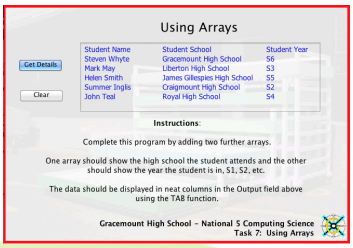
3rd Element of arrayBunk

Think of an Array as being a bit like a bunk bed

Getting to Grips with Arrays

Task 7: Using Arrays

Specification



Take the program above further by expanding it by not only asking for five **student names** but also their **school** attended and **year** group.

This details entered should then be displayed in neat columns in the Output box. The **TAB function** will allow you to do this.

The design of this program is shown below to give you a hand. Look at this carefully before starting.

Design: Pseudocode for "Get Details" Button

HAGGIS Design Language

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

1. Setup Arrays And Variable
2. Get Student Details
3. Display Details

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

1. Setup Arrays And Variable
1.1 **SET** arrayName **TO** String ""
1.2 **SET** arraySchool **TO** String ""
1.3 **SET** arrayYear **TO** String ""
1.4 **SET** Number_Of_Students **TO** Integer 5

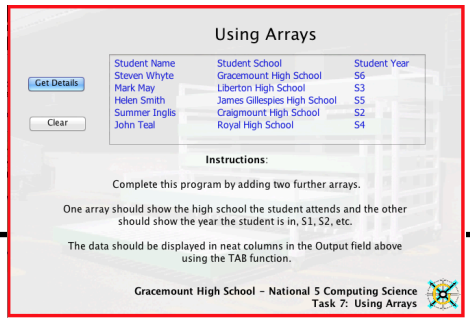
2. Get Student Details
2.1 **REPEAT** with loop = 1 to Number_Of_Students
2.2 **RECEIVE** arrayName[loop] **FROM (String) KEYBOARD**
2.3 **RECEIVE** arraySchool[loop] **FROM (String) KEYBOARD**
2.4 **RECEIVE** arrayYear[loop] **FROM (String) KEYBOARD**
2.5 **END REPEAT**

3. Display Details
3.1 **SEND** "Student Name" & **TAB** & "Student School" & **TAB** & "Student Year" **TO DISPLAY**
3.2 **REPEAT** with loop = 1 to Number_Of_Students
3.3 **SEND** arrayName[loop] & **TAB** & arraySchool[loop] & **TAB** & arrayYear[loop] **TO DISPLAY**
3.4 **END REPEAT**

Implementation

Open the "**Using Arrays**" stack. It can be found in:

N5 LiveCode Programming > 7_Using Arrays.livecode



Testing

Test your program once complete. Output should look similar to that shown above.

You can choose your own names, year groups and schools names.

Task 10: Youth Orchestra Program

Specification

A program is required to display the names, ages and two test marks out of **50** for pupils hoping to join the **junior** (**12-14** years) and **senior** (**15-17** years) orchestra. Both the **ages** and **marks** should be **validated**.

The **total** of the first and second marks should be worked out and a decision should be made as to whether or not they can join the **junior** or **senior orchestra** depending on their age.

If the students **total mark** is **over 70** and depending on their **age**, they are accepted to the orchestra. All of this information should be displayed in an Output field.

Design: Pseudocode for "Get Details" Button

HAGGIS Design Language

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

1. Setup variables
2. Get details
3. Make Decision
4. Display Data
5. Validate Mark

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

1. Setup variables
1.1 **SET** arrayName **TO** String ""
1.2 **SET** arrayAge **TO** Integer 0
1.3 **SET** arrayMark1 **TO** Integer 0
1.4 **SET** arrayMark2 **TO** Integer 0
1.5 **SET** arrayTotal **TO** Integer 0
1.6 **SET** arrayDecision **TO** String ""
1.7 **SET** Number_Of_Students **TO** Integer 4
1.8 **SET** Check_Mark **TO** Integer 0
1.9 **SET** Check_Age **TO** Integer 0

2. Get details
2.1 **REPEAT** with loop = 1 TO Number_Of_Students
2.2 **RECEIVE** arrayName[loop] **FROM (String) KEYBOARD**
2.3 **RECEIVE** Check_Age **FROM (Integer) KEYBOARD**
2.4 **REPEAT UNTIL** Check_Age is between 12 and 17 **AND** Check_Age is an integer
2.5 **SEND** error message if an invalid age is entered **TO DISPLAY**
2.6 **IF** the cancel button is pressed **THEN** exit to the top of the program
2.7 **RECEIVE** Check_Age **FROM (Integer) KEYBOARD**
2.8 **END REPEAT**
2.9 **SET** arrayAge[loop] **TO** Check_Age
2.10 **RECEIVE** Check_Mark **FROM (Integer) KEYBOARD**

The design is continued on the next page

Task 8: Youth Orchestra Program

2.11 **CALL** the Validate Mark function
2.12 **SET** arrayMark1[loop] **TO** Check_Mark once validated
2.13 **RECEIVE** Check_Mark **FROM (Integer) KEYBOARD**
2.14 **CALL** the Validate Mark function
2.15 **SET** arrayMark2[loop] **TO** Check_Mark once validated
2.16 **SET** arrayTotal[loop] **TO** arrayMark1[loop] + arrayMark2[loop]
2.17 **END REPEAT**

3. Make Decision
3.1 **REPEAT** with loop = 1 TO Number_Of_Students
3.2 **IF** arrayAge[loop] is between >=12 **AND** <=14 **AND** arrayTotal[loop] > 70 **THEN SET** arrayDecision[loop] **TO** "You are accepted to junior orchestra"
3.3 **IF** arrayAge[loop] is between >=15 **AND** <=17 **AND** arrayTotal[loop] > 70 **THEN SET** arrayDecision[loop] **TO** "You are accepted to the senior orchestra"
3.4 **IF** arrayTotal[loop] <= 70 **THEN SET** arrayDecision[loop] **TO** "You have been declined to the orchestra."
3.5 **END REPEAT**

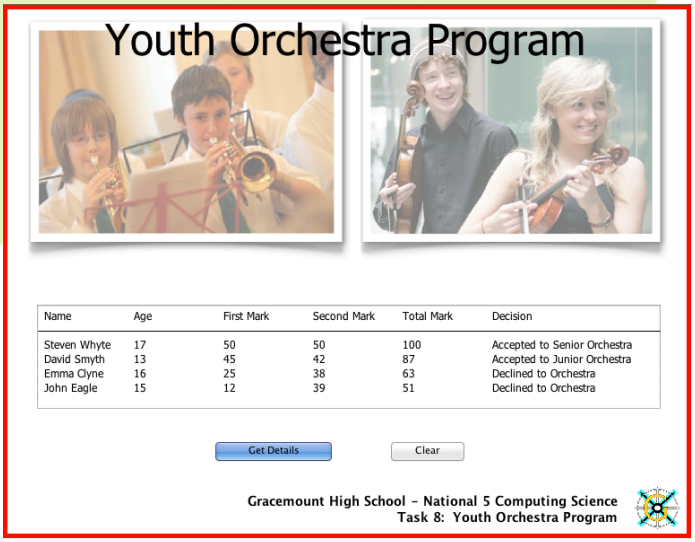
4. Display Data
4.1 **SEND** "Name" & **TAB** & "Age" & **TAB** & " First Mark" & **TAB** & "Second Mark" & **TAB** & "Total Mark" & **TAB** & "Decision" **TO DISPLAY**
4.2 **REPEAT** with loop = 1 TO Number_Of_Students
4.3 **SEND** arrayName[loop] & **TAB** & arrayAge[loop] & **TAB** & arrayMark1[loop] & **TAB** & arrayMark2[loop] & **TAB** & arrayTotal[loop] & **TAB** & arrayDecision[loop] **TO DISPLAY**
4.4 **END REPEAT**

5. Validate Mark
5.1 **REPEAT UNTIL** Check_Mark >= 1 **AND** Check_Mark <=50 **AND** Check_Mark is an Integer
5.2 **SEND** a message telling the user that they have entered an invalid mark **TO DISPLAY**
5.3 **IF** the cancel button is pressed **THEN** exit to the top of the program
5.4 **RECEIVE** Check_Mark **FROM (Integer) KEYBOARD**
5.5 **END REPEAT**

Implementation

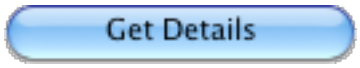
Once you have read the design above, open the "**Youth Orchestra Program**" stack. It can be found in:

N5 LiveCode Programming >
8_Youth Orchestra Program.livecode



Task 8: Youth Orchestra Program

Implementation (continued)



Assign the following script to the **Get Details** button.

```
// Setup the global arrays and variables to be used in this event
Global arrayName, arrayAge, arrayMark1, arrayMark2, arrayTotal, arrayDecision, Number_Of_Students,
Check_Mark, Check_Age

On MouseUp
  Setup_Arrays_And_Variables
  Get_Details
  Make_Decision
  Display_Data
End MouseUp

On Setup_Arrays_And_Variables
  // Initialise the global variables and arrays
 Put "" into arrayName
 Put 0 into arrayAge
 Put 0 into arrayMark1
 Put 0 into arrayMark2
 Put 0 into arrayTotal
 Put "" into arrayDecision
 Put 4 into Number_Of_Students
 Put 0 into Check_Mark
 Put 0 into Check_Age
End Setup_Arrays_And_Variables

On Get_Details
 // Setup the number of students - this will determine how many times the loop repeats
  // Start a fixed loop and ask for the students name, age, and first and second mark
 REPEAT with loop = 1 to Number_Of_Students
    Ask "Please enter the name of student number " & loop
    Put it into arrayName[loop]
    //
    Ask "Please enter the age of " & arrayName[loop]
    Put it into Check_Age
    //
    // Check that the age entered is between 12 and 17 and is a whole number (integer).
    // If the age entered is not in this range or a whole number, the user is asked to re-enter.
    REPEAT until Check_Age >= 12 and Check_Age <= 17 and Check_Age is an integer
      Ask "You have entered an invalid age. Please re-enter an age between 12 and 17"
      // 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 Check_Age
    END REPEAT
    Put Check_Age into arrayAge[loop]
    //
    Ask "Please enter the first mark for" &&arrayName[loop]
    Put it into Check_Mark
    Validate_Mark
    Put Check_Mark into arrayMark1[loop]
    //
    Ask "Please enter the second mark for" &&arrayName[loop]
    Put it into Check_Mark
    Validate_Mark
    Put Check_Mark into arrayMark2[loop]
    Put arrayMark1[loop] + arrayMark2[loop] into arrayTotal[loop]
  END REPEAT
End Get_Details
```

The code is continued on the next page

Task 8: Youth Orchestra Program

```
On Make_Decision
  //  Start a fixed loop
  REPEAT with loop = 1 to Number_Of_Students
    //  Use a series of conditional IF statements to determine whether or not the student has been
    //  accepted into the orchestra
    IF arrayAge[loop] >= 12 AND arrayAge[loop] <= 14 AND arrayTotal[loop] > 70 THEN put "Accepted
    to Junior Orchestra" into arrayDecision[loop]  //  On the same line
    IF arrayAge[loop] >= 15 AND arrayAge[loop] <= 17 AND arrayTotal[loop] > 70 THEN put "Accepted
    to Senior Orchestra" into arrayDecision[loop]  //  On the same line
    IF arrayTotal[loop] <= 70 THEN put "Declined to Orchestra" into arrayDecision[loop]
  END REPEAT
End Make_Decision

On Display_Data
  //  Display the column headings
  Put "Name" & tab  & "Age" & tab & "First Mark" & tab & "Second Mark" & tab & "Total Mark" & tab &
  "Decision" into line 1 of field "Output"  //  On the same line

  //  Start a fixed loop which will display all of the data contained within the arrays
  REPEAT with loop = 1 to Number_Of_Students
    Put arrayName[loop] & tab & arrayAge[loop] & tab & arrayMark1[loop] & tab & arrayMark2[loop] & tab
    & arrayTotal[loop] & tab & arrayDecision[loop] into line loop+3 of field "Output"  //  On the same line
  END REPEAT
End Display_Data


On Validate_Mark
  //  Check that the marks entered are between 0 and 50 and are whole numbers (integers).
  //  If the marks entered are not in this range or not whole numbers, the user is asked to re-enter.
  REPEAT until Check_Mark >= 0 and Check_Mark <= 50 and Check_Mark is an integer
    Ask "You have entered an invalid mark.  Please re-enter an mark between 0 and 50"
    IF the result = "Cancel" THEN exit to top
    Put it into Check_Mark
  END REPEAT
End Validate_Mark
```

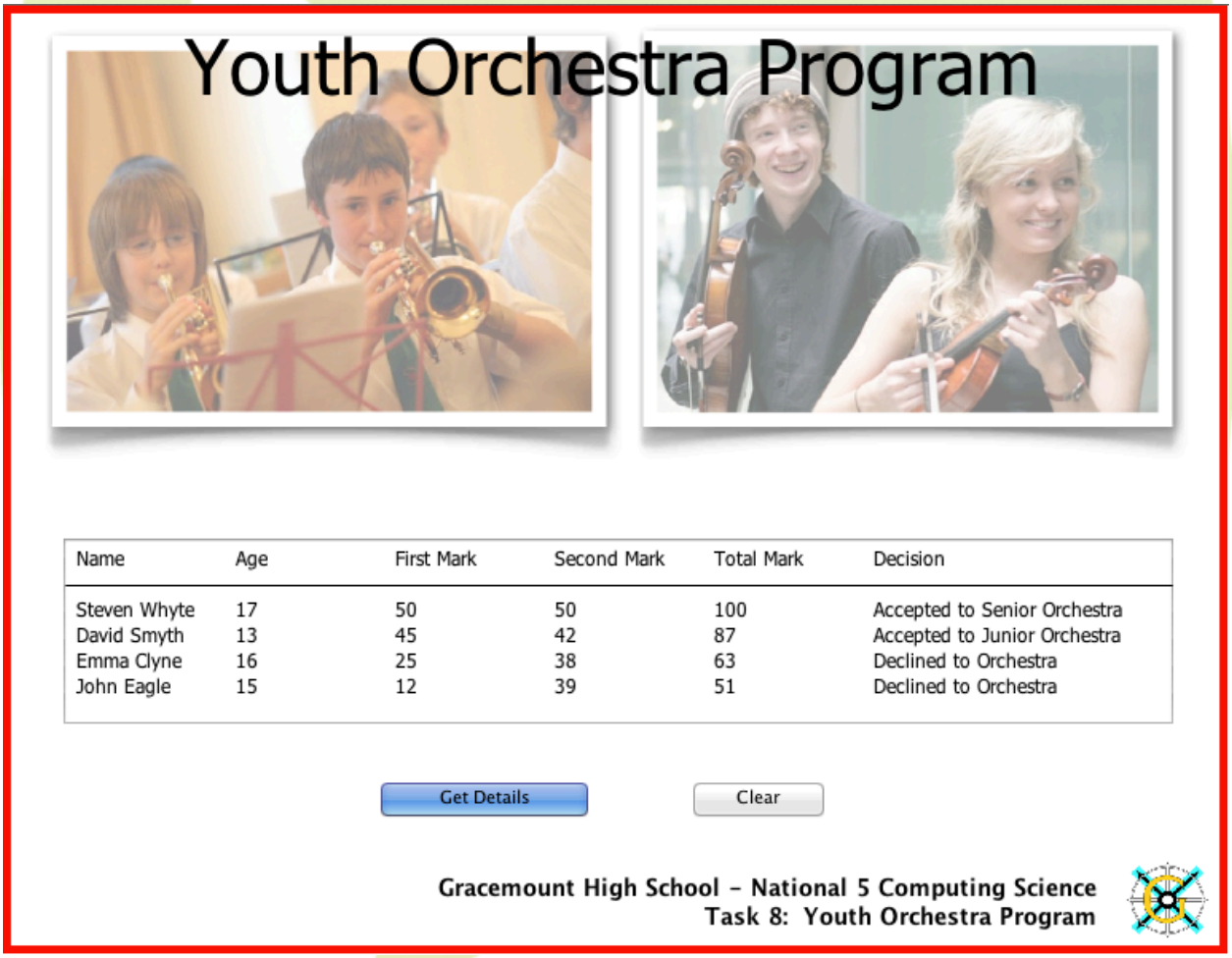
Task 8: Youth Orchestra Program

Testing

Test that you program produces the correct results by keying in the following test data in the screenshot below.

Note. When testing your program, be sure to **check** that the **valid mark** and **age** functions are **working correctly**. Remember:

- All **ages** that are keyed in must be **between 12 and 17**.
- All **marks** that are keyed in must be **between 0 and 50**.
- A student should be **accepted** to the **junior orchestra** if the **age** is **between 12 and 14** and their **total mark** is **over** (>) **70**.
- A student should be **accepted** to the **senior orchestra** if the **age** is **between 15 and 17** and their **total mark** is **over** (>) **70**.
- A student should be **declined** entry to the orchestra if their **total mark** is **less** than or **equal** to (<=) **70**.



Task 9: Go Ape!

Specification

GoApe is wanting to create an program that would calculate the cost for a group to visit their high ropes assault courses

This program will have a choice of 4 different activities:

- **Baboon Tree Top Adventure - £24 per person**
- **Gorillas Tree Top Adventure - £32 per person**
- **Forest Segway - £30 per person**
- **Tree Top Junior - £16 per person**

The Program will let the user select how many people will be participating in each activity. It will also ask them for one contact name and date of arrival and will then calculate the total cost and add on VAT (20% extra).

Design: Pseudocode for "Calculate Price" Button

HAGGIS Design Language

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

1. Setup Variables
2. Collect Info

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

1. Setup Array and Variables
1.1 **SET** YourName **TO** String ""
1.2 **SET** YourDateOfArrival **TO** String ""
1.3 **SET** TotalCost **TO** Real 0.00
1.4 **SET** VAT **TO** Real 0.00

2. Collect Information and Display
2.1 **IF** arrayOfActivities is empty **THEN**
2.2 **SEND** "No activities selected" **TO DISPLAY**
2.3 **ELSE**
2.4 **RECEIVE** YourName **FROM (String) KEYBOARD**
2.5 **RECEIVE** YourDateOfArrival **FROM (String) KEYBOARD**
2.6 **SEND** "Thank you for your enquiry " & YourName **TO DISPLAY**
2.7 **SEND** "Arrival date: " &YourDateOfArrival **TO DISPLAY**
2.8 **SET** TotalCost **TO** 24 * arrayOfActivities[1] + 32 * arrayOfActivities[2] + 30 * arrayOfActivities[3] + 16* arrayOfActivities[4]
2.9 **SET** VAT **TO** 0.2 * TotalCost
2.10 **SET** Number Format **TO** "0.00"
2.11 **SEND** TotalCost + VAT **TO DISPLAY**
2.12 **END IF**

Task 9: Go Ape!

Implementation

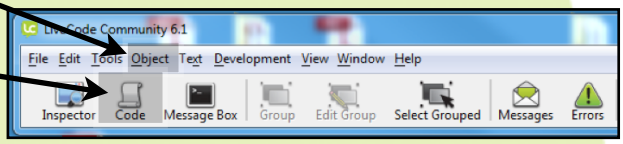
Once you have read the design above, open the "**Go Ape**" stack. It can be found in:

N5 LiveCode Programming > 9_Go Ape.livecode

The main card requires this code so that when the programs starts it initialises the array and sets all the drop down menus to "Select"

This code can be entered by pressing **Object** option and then **Card Inspector**.

Then press the **Code** icon.



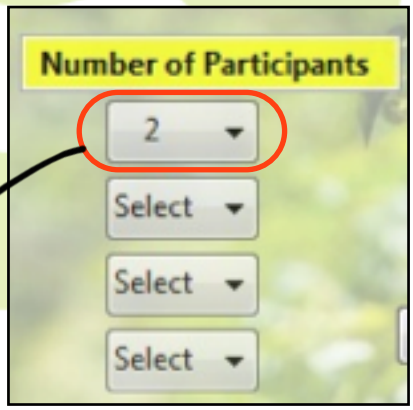
Enter the following code into the script window.

```
Global arrayOfActivities

On OpenCard
  Setup_Card
End OpenCard

On Setup_Card
  Put empty into field "Output"
  Put empty into arrayOfActivities

  Set the label of button "BaboonOption" to "Select"
  Set the label of button "GorillaOption" to "Select"
  Set the label of button "BaboonSegway" to "Select"
  Set the label of button "JuniorOption" to "Select"
End Setup_Card
```

Implementation (continued)

Each drop down menu button requires code **like this** to acquire the number of people doing each activity.

<u>The code will be slightly different for each menu though!</u>



Drop Down Menus

```
Global arrayOfActivities

// This section of code transfers the information from the drop
// down menu button into the array

On menuPick ItemPicked

// When you select a number from the drop down menu this is
// transferred into ItemPicked so if you picked 4 people the
// ItemPicked would become 4

  Answer "You selected "& ItemPicked &" people to do Tree Top Adventure Baboon"

  // Store this number into the array
  Put ItemPicked into arrayOfActivities[1]

End menuPick
```

Task 9: Go Ape!

Implementation (continued)

This is the code that will go into the “Calculate Price” button.
Your teacher will go over this with you, **make sure you understand it**.

```
Global arrayOfActivities, YourName, YourDateOfArrival, TotalCost, VAT

On mouseUp
  Setup_Variables
  Collect_Info
End mouseUp

On Setup_Variables

  // Setup the variables to be used in this event
  Put "" into YourName
  Put "" into YourDateOfArrival
  Put 0.00 into TotalCost
  Put 0.00 into VAT

End Setup_Variables

On Collect_Info

  // Check it see if any information has been put into the array
  IF arrayOfActivities is empty THEN
    Answer "Please select some activities before you select the Calculate Price button"
  ELSE
    Ask "Please enter you name: "
    Put it into YourName

    Ask "Please enter date of arrival: "
    Put it into YourDateOfArrival

    Put "Thank you for your enquiry " & YourName into line 1 of field "Output"
    Put "Arrival date: " &YourDateOfArrival into line 2 of field "Output"

    // Each element from arrayOfActivites is multiplied by the price for that activity
    Put 24 * arrayOfActivities[1] + 32 * arrayOfActivities[2]+ 30 * arrayOfActivities[3]+16 *
    arrayOfActivities[4] into TotalCost // On the same line

    // Calculate 20% of the total bill in order to work out the VAT cost
    Put 0.2 * TotalCost into VAT

    Set numberformat to "0.00"
    Put "Cost: £" & TotalCost into line 3 of field "Output"
    Put "VAT: £" & VAT into line 4 of field "Output"
    Put "Total cost: £" & TotalCost + VAT into line 5 of field "Output"
  END IF

End Collect_Info
```

Task 9: Go Ape!

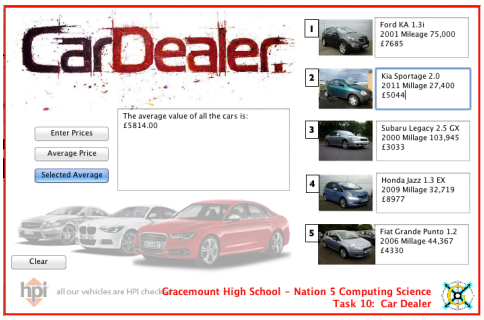
Testing

You should now test that your program works correctly. Run your program twice with the data in the test table below. Ensure that the **Cost**, **VAT** and **Overall Cost** are calculated correctly.

Activities on Offer	Cost Per Activity (£)	Number Of Participants In Each Activity	Calculated Total Pay	Programs Total Pay
Program Run 1: **Customer Name: Mr Steven Whyte**			**Date of Arrival: 10th September 2014**	
1. Tree Top Adventure Baboon	£24.00	6	Cost: £640.00	Cost: £640.00
2. Tree Top Adventure Gorillas	£32.00	5	VAT: £128.00	VAT: £128.00
3. Forrest Segway	30.00	8	Overall Cost: £768.00	Overall Cost: £768.00
4. Tree Top Junior	£16.00	6		
Program Run 2: **Customer Name: Miss Emily Smyth**			**Date of Arrival: 21st May 2014**	
1. Tree Top Adventure Baboon	£24.00	9	Cost: £918.00	Cost: £918.00
2. Tree Top Adventure Gorillas	£32.00	9	VAT: £183.60	VAT: £183.60
3. Forrest Segway	30.00	9	Overall Cost: £1101.60	Overall Cost: £1101.60
4. Tree Top Junior	£16.00	9		

Task 10: Car Dealer

Specification



A car dealer requires a program in order to advertise and sell his top five vehicles each week.

The program should take in the **cost** of the **five cars** and then allow the user to display the **average** of all the car prices entered.

It should also allow the user to **work out** the **average** of a **range of cars** entered and not just all of them.

For example, the car dealer should be able to use the program to **work out** and display the **average** of cars 1, 3 and 5 if need be.

Design: Pseudocode for "Enter Prices" Button

HAGGIS Design Language

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

1. Setup Variables And Array
2. Get Prices
3. Validate Price

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

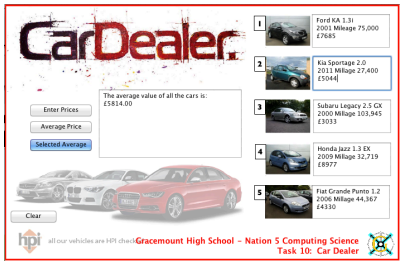
1. Setup Variables And Array
1.1 **SET** CarValue **TO** Real 0.00
1.2 **SET** arrayCarPrices **TO** Real 0.00
1.3 **SET** NumberOfCars **TO** Integer 5

2. Get Prices
2.1 **REPEAT** with loop = 1 to NumberOfCars
2.2 **RECEIVE** CarValue **FROM (real) KEYBOARD**
2.3 **CALL** Validate_Price Function
2.4 **SET** CarValue **TO** arrayCarPrices[loop]
2.5 **SEND** "Car Output " & loop **TO** Field_Name
2.6 **SEND** "£" & arrayCarPrices[loop] into line 3 of field Field_Name
2.7 **END REPEAT**

3. Validate Price
3.1 **REPEAT** until CarValue > 0
3.2 **RECEIVE** CarValue **FROM (real) KEYBOARD**
3.3 **END REPEAT**

Task 10: Car Dealer

Implementation



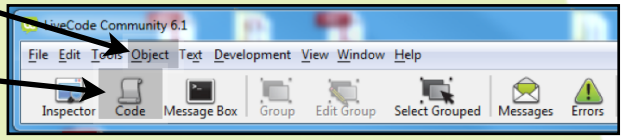
Once you have read the design above, open the "**Car Dealer**" stack. It can be found in:

N5 LiveCode Programming > 10_Car Dealer.livecode

The main card requires this code so that when the program starts it empties (clears) line 3 of each of the prices in each of the car Output fields.

This code can be entered by pressing **Object** option and then **Card Inspector**.

Then press the **Code** icon.



Enter the following code into the script window.

```
Global NumberOfCars, arrayCarPrices

On OpenCard
  Setup_Card
End OpenCard

On Setup_Card
  //  Set up the variable and array to be used in this event
  Put 5 into NumberOfCars
  Put 0 into arrayCarPrices

  //  Clear line 3 from each of the 5 Car Output fields
  REPEAT with loop = 1 to NumberOfCars
    Put "Car Output " & loop into Field_Name
    Put empty into line 3 of field Field_Name
  END REPEAT

  //  Clear the Average Output field
  Put empty into field "Average Output"
End Setup_Card
```

Implementation (continued)

The following code should now be entered into the **"Enter Prices"** button.

```
Global arrayCarPrices, NumberOfCars, CarValue

On mouseUp
  Setup_Variables_And_Array
  Get_Prices
End mouseUp

On Setup_Variables_And_Array
  //  Select the variables and array to be used in this event
  Put 0 into CarValue
  Put 0 into arrayCarPrices
  Put 5 into NumberOfCars
End Setup_Variables_And_Array
```

The code is continued on the next page

Task 10: Car Dealer

```
On Get_Prices
  //  Start a Repeat until loop in order to get the cost of each of the cars
  REPEAT with loop = 1 to NumberOfCars
    Ask "Enter the value of Car "&loop&" to the nearest pound (don't put in the £ sign)"
    IF the result = "Cancel" THEN exit to top
    Put it into CarValue

    //  Begin the Validate_Price function
    Validate_Price

    //  Put each price entered into each of the five elements of the arrayCarPrices
    Put CarValue into arrayCarPrices[loop]

    //  Put the price into line 3 of each of the Car Output fields
    Put "Car Output " & loop into Field_Name
    Put "£" & arrayCarPrices[loop] into line 3 of field Field_Name
  END REPEAT
End Get_Prices

On Validate_Price
  //  Each price must be greater than £0.00.  The validation function will check this
  REPEAT until CarValue > 0
    Ask "Car value must be greater than £0.00"
    IF the result = "Cancel" THEN exit to top
    Put it into CarValue
  END REPEAT
End Validate_Price
```

Implementation (continued)

The following code should now be entered into the **"Average Price"** button.

```
Global arrayCarPrices, NumberOfCars, TotalCarValue, AverageCarValue

On mouseUp
  // This boolean statement checks to see that car prices have been entered
  IF arrayCarPrices = 0 THEN
    Answer "Please enter car prices before selecting the Average Price Button"
  ELSE
    Setup_Variables
    Display_Average
  END IF
End mouseUp

On Setup_Variables
  //  Setup the variables to be used in this event
  Put 0.00 into TotalCarValue
  Put 0.00 into AverageCarValue
End Setup_Variables
```

The code is continued on the next page

Task 10: Car Dealer

```
On Display_Average

  //  Put each of the car prices into a variable to hold the TotalCarValue of all the cars
  REPEAT with loop = 1 to NumberOfCars
    Put arrayCarPrices[loop] + TotalCarValue into TotalCarValue
  END REPEAT

  //  Work out the average by dividing the TotalCarValue by the NumberOfCars and
  //  round the answer.  Place the answer into the variable AverageCarValue
  Put round(TotalCarValue / NumberOfCars) into AverageCarValue

  //  Set the Output to pounds and pence
  //  Display the AverageCarValue
  Set numberformat to "0.00"
  Put "The average value of all the cars is: £" & AverageCarValue into field "Average Output"
End Display_Average
```

Implementation (continued)

The following code should now be entered into the **"Selected Average"** button.

```
Global arrayCarPrices, CarPricesEntered, RunningTotal, NoMoreCars, NumberOfCarsEntered,
carNumber // On the same line

On mouseUp
  // This boolean statement checks to see that car prices have been entered
  IF arrayCarPrices = 0 THEN
    Answer "Please enter the car prices before selecting the Selected Average Button"
  ELSE
    Setup_Variables
    Select_Range
  END IF
End mouseUp

On Setup_Variables
  //  Setup the variables to be used in this event
  Put 0 into carNumber
  Put "" into NoMoreCars
  Put 0.00 into RunningTotal
  Put 0 into NumberOfCarsEntered
End Setup_Variables

On Select_Range
  //  Allow the user to choose the cars that they want to find the average for.
  REPEAT until NoMoreCars = "N"
    Ask "Enter the cars that you want to work out the average for, enter N when finished: "
    IF the result = "Cancel" THEN exit to top
    Put it into carNumber

    //  Every time user user chooses a car, one is added to the NumberOfCarsEntered
    IF carNumber <> "N" THEN
      Add 1 to NumberOfCarsEntered
```

The code is continued on the next page

Task 10: Car Dealer

```
      // Check to see that a valid car number between 1 and 5 has been entered
      REPEAT until carNumber >= 1 AND carNumber <= 5 AND carNumber is an integer
        Ask "You have to select a car between 1 and 5"
        IF the result = "Cancel" THEN exit to top
        put it into carNumber
      END REPEAT
      Put arrayCarPrices[carNumber]+RunningTotal into RunningTotal
    ELSE
      // If the "No" is entered, come out of the Repeat until loop
      Put it into NoMoreCars
    END IF
  END REPEAT

  // Set the Output to pounds and pence
  // Calculate the selected average by dividing the RunningTotal by the
  // NumberOfCarsEntered by the user
  // Display the result in the Average Output field
  Set numberformat to "0.00"
  Put "The average cost for the cars you selected is £"&RunningTotal/NumberOfCarsEntered
  into field "Average Output"  // On the same line
End Select_Range
```

Testing

You should now test that your program works correctly using the test data supplied on the next page.

Task 10: Car Dealer

Testing

You should now test that your program works correctly. Run your program twice with the data in the test table below. Ensure that the **Average** and **Selective Average** on the range of cars shown are used, and are calculated correctly.

Cars For Sale	Cost Per Car (£)	Calculated Average	Programs Average	Calculated Selective Average	Programs Selective Average
Program Run:					
1. Ford KA	£7500				
2. Kia Sportage	£4300			Cars Entered: 1, 3 and 5	Cars Entered: 1, 3 and 5
3. Subaru Legacy	£5000	£6,240.00	£6,240.00	£5766.67	£5766.67
4. Honda Jazz	£9600				
5. Fiat Grande Punto	£4800				

You are now ready to do the End of Unit Outcome Assessments for Software Design and Development Topic.

Good luck!