

Portfolio Tasks

1.1T □ Hello World

Write a program in Ruby that outputs `Hello World`.

Click on 'Terminal' then type **`ruby hello.rb`** to run your code. When you are ready click 'Mark' to see if your program works as expected.

Operators Quiz

Question 1

Assigning a value to a variable is done using the `=` operator

☐ True

☐ False

Question 2

The expression `a + b` is **evaluated** to find the sum.

☐ True

☐ False

Question 3

The following statements can be executed in any order:

```
1  a = 3
2  b = 4
3  a = b + c
4  c = a
5  puts(" c has the value #{c}")
```

☐ True

☐ False

Testing Programs

The following is an example of how to test a program - it shows the **test data** and the **expected result** .

We want to make sure that when we run our program the **actual output** matches the **expected result**.

The **pseudocode** is a description of what the program does in regular English (i.e no particular programming language):

The **required variables** describes where data is stored in the program, in this case the data being stored is **Integer** data (i.e whole numbers):

Program 1: Add 2 numbers (*Example of Desk Checking*)

Required Variables:

Integer: a, b, c.

Pseudocode:

Read the value of **a**

Read the value of **b**

Add **a** to **b** and assign the result to **c**

Print the value of **c** to the terminal.

Test Data:

	<i>First data set</i>	<i>Second data set</i>
a	10	3
b	5	4

Expected Result:

	<i>First data set</i>	<i>Second data set</i>
Output:	15	7

Run the Bill Total program

Complete and run the program Bill Total in the Terminal using the test data below.

Record the actual output of the program using this data and check it matches the expected result.

Test Data:

	<i>First data set</i>	<i>Second data set</i>
appetizer_price	10.30	12.40
main_price	34.00	41.00
dessert_price	8.50	9.80

Expected Result:

	<i>First data set</i>	<i>Second data set</i>
Output:	\$52.80	\$63.20

If your program is working as expected click the 'Mark' button and make sure your program passes the tests.

1.2T M ☐☐ Desk Check the Bill Total program

Complete the attached document - filling in the fields based on the output of your program. Check it is working correctly, then answer the questions.

Upload your completed document (AS A PDF) to the workspace.



[Tutorial Task 1.2 - Answers.docx](#)

Click the 'Mark' button to submit your task for manual marking by your tutor.

Reading and Writing Using Variables

Variables allow you to store values that can change in your program.

When you declare a **local variable** you are telling the computer to create a variable for use within the function or procedure.

You must give the variable a name and keep in mind the type of data it will store. The computer will then set aside space for you to store a value for that variable.

You can then write values to the variable and read values back.

▶ Run

RUBY



```
1 # This is a variable called x we assign it the value 10
2 x = 10
3 # Print out the value of x:
4 puts("x is #{x}")
5 # read in another value for x:
6 puts("Enter an integer value other than 10: ")
7 x = gets().to_i()
8 # print the new value of x:
9 print("x is now #{x}")
```



1.3P □ Hello User

Change the code provided to:

1. Read the user's name (a String, prompt with 'Please enter your name: ') and store it in the name variable.
2. Print out the user's name followed by an exclamation mark.
3. Read the user's family name, remove any white space and store it in the family_name variable.
4. Print out the user's family_name followed by an exclamation mark.
5. Read in the user's year of birth
6. Convert the year of birth to an integer then calculate the user's age and print it out.
7. Prompt for and read in the user's height in metres
8. Convert the height to inches and print out the result.


2.1T □ Debug coding task

Fix the code so that it works correctly. Sample output is as follows (***you may need to see the hints on the next slide to help you***)

```
Enter your age in years:
11
Enter your name:
Jasper
Jasper you were born in: 2008
```

Remember:

1. `.to_s` – converts a type to a string.
2. `.to_i` – converts an integer in string form to numeric form.
3. `require 'date'` – includes the date library.
4. `.chomp` – removes whitespace from a string (eg: newlines, tabs, spaces).

 Look carefully at the comments in the code - they will help you!

Hints for fixing the debug code

Example of how to fix the program:

We are looking for the **where** and the **what**.

The first error message you encounter looks as follows:

```
MacBook-Pro-6:Resources mmitchell$ ruby debug.rb
Enter your age in years:
11
debug.rb:10:in `get_age': undefined local variable or method `age_in_years' for main:Object (NameError)
    from debug.rb:29:in `main'
    from debug.rb:34:in `<main>'
MacBook-Pro-6:Resources mmitchell$
```

The part that says: `debug.rb:10:in `get_age':` indicates both the line and the function **where** the error occurred, in this case line 10 which is in the function `get_age()`.

The part that says: `undefined local variable or method `age_in_years'` indicates the **what** – in this case the code refers to something called `age_in_years` which we have not defined as either a variable or a function – so Ruby does not know what it is.

So how do we fix this problem?

```
# Asks the user to enter their age and returns an integer age
def get_age()
  puts "Enter your age in years: "
  age = gets
  return age_in_years
end
```

This variable `age` should be called `age_in_years`

Without the change in name (see box above) this variable has not been declared.

So now you need to continue with the other errors that arise.

2.2T □ Hello User with Functions

In this task you need to use the functions in the library `input_functions.rb` to create a simple interactive program. This is a variation on your previous Hello User task.

The output should look as follows:

```
What is your name?  
Sam  
Your name is Sam!  
What is your family name?  
McClan  
Your family name is: McClan!  
What year were you born?  
2012  
So you are 7 years old  
Enter your height in metres (i.e as a float):  
1.1  
Your height in inches is:  
43.30711  
Finished  
Do you want to continue?  
no  
ok, goodbye
```

The steps needed are:

Use the functions in `input_functions.rb` to do the following:

1. Read the user's name (a `String`, prompt with 'Please enter your name: ') and store it in the `name` variable.
2. Print out the user's name followed by an exclamation mark.
3. Read the user's family name and store it in the `family_name` variable.
4. Print out the user's family_name followed by an exclamation mark.
5. Read in the user's year of birth as an integer
6. Calculate the user's age and print it out.
7. Prompt for and read in the user's height in metres as a float
8. Convert the height to inches and print out the result. Use the constant `INCHES` at the top of the code to convert metres to inches
9. Call the `read_boolean` function from `input_functions.rb` passing in the argument 'Do you want to continue?'
10. If the `read_boolean` function returns true then display 'Ok, lets continue ', if the function returns false then output 'ok, goodbye.'

This should look as follows:

RUBY



```
1 continue = read_boolean('Do you want to continue?')
2 if (continue)
3     # ok message here
4 else
5     # good bye message here
6 end
```



2.3P □ Write your own functions (Hospital Charges)

Complete the program below so that it produces the correct output - for example:

```
Enter patient name:
Sam Jones
Enter the accommodation charges:
23.50
Enter the theatre charges:
45.99
Enter the pathology charges:
59.20
The patient name: Sam Jones
The total amount due is: $128.69
```

Steps in the program:

1. Read in the patient's name (you need to write this - use the `read_string(prompt)` function from the file `input_functions.rb`).

*NOTE: **you do not need** to copy in the functions from `input_functions.rb` - they are available because of the*

'require './input_functions.rb' line at the top of the program.

2. Calculate the accommodation charges (these are just read in)

3. Calculate the theatre charges (these are just read in)

4. Read in the pathology charges (you need to write this to read the in)

5. Print to the terminal the patient's name and the bill total (using the function `print_float(value, decimal_places)` from the file `input_functions.rb`).

□ Types Quiz

Reorder the Data Types to match the Field Names below:

Question

1. Name
2. Family Name
3. Year Born
4. Date
5. Age
6. Height
7. Continue

NB: Some types have (1) or (2) after them - use these in that order.

Boolean

Float

String (1)

String (2)

Integer (1)

Integer (2)

Date

3.1.1T □ Name Tester - first stage

Implement a main procedure with the following logic:

- It reads a name from the user, and displays back a message.
- Check if the name entered is your name or your tutor's name.
- If the name is your name or your tutor's name output the message: '#{name} is an awesome name!'
- Otherwise output the silly name message: '#{name} is a silly name

Eg:

```
What is your name?  
Ted  
Ted is an awesome_name!
```

And:

```
What is your name?  
JillyJenny  
JillyJenny is a silly name
```

3.1.2T □ Name Tester - second stage

In this challenge we add a procedure with a loop to our Name Tester code so as to produce the following output when the name is not awesome:

```
What is your name?  
Sam  
Sam is a  
silly silly silly silly silly silly silly silly silly silly silly silly si  
lly silly silly silly silly silly silly silly silly silly silly silly sill  
y silly silly silly silly silly silly silly silly silly silly silly silly  
silly silly silly silly silly silly silly silly silly silly silly silly si  
lly silly silly silly silly silly silly silly silly silly silly silly si
```

The pseudocode for this is as follows:

```
Make i equal 0  
Print (staying on the same line) name, ' is a'  
While i is less than 60  
do  
    Print (on the same line) 'silly '  
    Increment i (make i equal i + 1)  
end  
Output ' name!' (moving to a new line)
```

3.2T M □ Simple Menu Program

Modify a small program that will give the user the following options:

1. Enter or Update an album
2. Play an Existing album
3. Exit the system

The sample output below shows that when run your program should look as follows:

```
Main Menu:
1 To Enter or Update Album
2 To Play Existing Album
3 Exit
Please enter your choice:
1
Maintain Albums Menu:
1 To Update Album Title
2 To Update Album Genre
3 To Enter Album
4 Exit
Please enter your choice:
1
You selected Update Album. Press enter to continue

Maintain Albums Menu:
1 To Update Album Title
2 To Update Album Genre
3 To Enter Album
4 Exit
Please enter your choice:
2
You selected Update Album Genre. Press enter to continue

Maintain Albums Menu:
1 To Update Album Title
2 To Update Album Genre
3 To Enter Album
4 Exit
Please enter your choice:
3
You selected Enter Album. Press enter to continue

Maintain Albums Menu:
1 To Update Album Title
2 To Update Album Genre
3 To Enter Album
4 Exit
Please enter your choice:
4
Main Menu:
1 To Enter or Update Album
2 To Play Existing Album
3 Exit
Please enter your choice:
2
You selected Play Existing Album. Press enter to continue
█
```

Notice that for each option that does not lead to a menu you need to implement a **stub** - i.e code that is just there to test the menu and does not have the final and full functionality yet. In the sample code there is a stub for Main Menu option 2: Play existing Album - you need to implement the others.

□ GOSU Resources

Sobkowicz, M 2015 *Learn game programming with Ruby : bring your ideas to life with Gosu*, The Pragmatic Bookshelf (See chapter 7 for help the grid aspect of the Maze Task)

[Gosu Ruby Documentation](#)

[Gosu site](#)

[Gosu game video tutorial](#)

3.3C M □ Drawing Shapes using GOSU

In this task you use Gosu to create a program that draws a picture.

GOSU is a development environment that makes it easy to create programs that use graphics, sounds, animations and other aspects relevant to creating small interactive games.

Follow these **3** steps:

1. Copy the code provided to your IDE (both `gosu_shapes.rb` and `circle.rb`) and use the demonstration shapes to create a picture of your own design. Your picture should include at least 3 different types of shapes (eg: a triangle, a rectangle and a circle)

Use the following site to select colours for the circle (which uses RGB values):

https://www.rapidtables.com/web/color/RGB_Color.html

Or Use the Gosu colour constants:

<https://www.rubydoc.info/github/gosu/gosu/master/Gosu/Color>

eg: a Red circle with a radius of 50 pixels would be produced by the two statements:

```
img = Gosu::Image.new(Circle.new(50))
img.draw(200, 200, ZOrder::TOP, 0.5, 1.0, Gosu::Color::RED)
```

Or you could use the HEX values:

```
img.draw(300, 50, ZOrder::TOP, 1.0, 1.0, 0xff_ff0000)
```

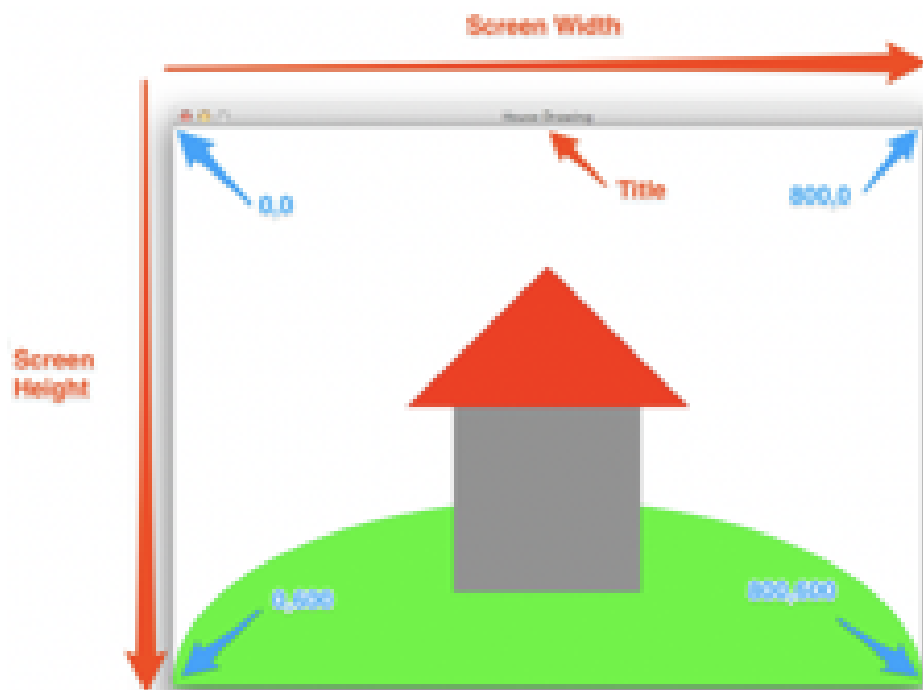
See here to work out HEX values:

<https://www.binaryhexconverter.com/decimal-to-hex-converter>

2. Run the code provided and study it to understand what is happening (you may need to run the command `gem install rubygems` to use the `circle.rb` code).

3. Using an IDE like Visual Studio Code, change the code to draw your own unique picture. You might want to draw it on paper first (perhaps graph paper or an electronic equivalent [like this](#)).

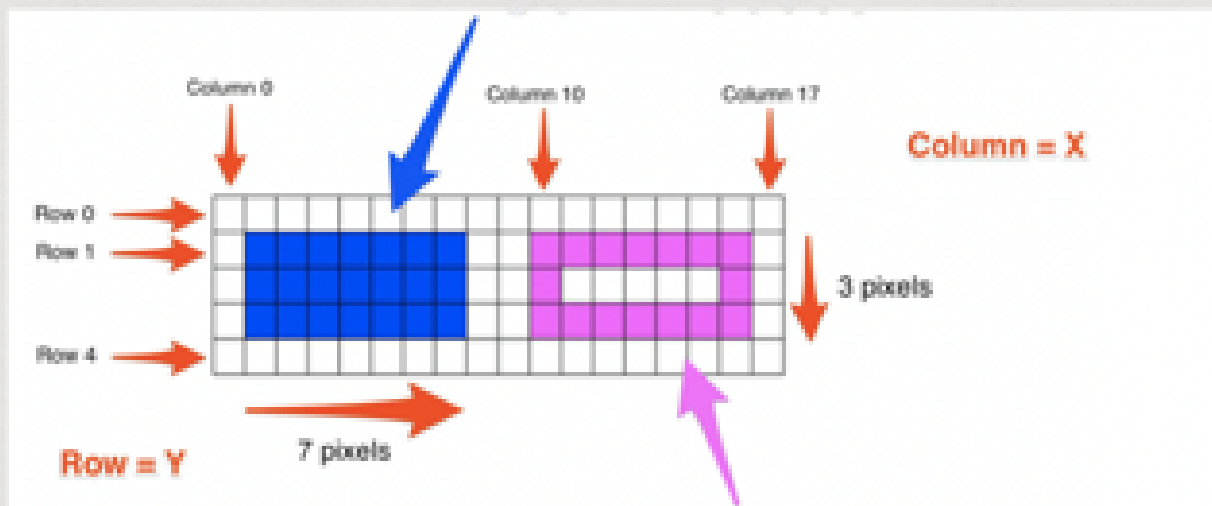
The co-ordinate system works as follows



Example:

Drawing a rectangle:

`Gosu.draw_rect(1, 1, 7, 3, Gosu::Color::BLUE, ZOrder::TOP, mode=:default)`



To draw this you need to draw 4 lines,
a good opportunity to write a
`draw_open_rectangle()` procedure!

You might want to first draw your shape on graph paper:

<https://print-graph-paper.com/virtual-graph-paper>

Once your code is complete submit a screen shot to the workspace.

4.1T □ File Handling

Files allow you to store data persistently. In this task you will write a simple file reading program to read multiple lines using a loop printing each line read to the terminal screen.

To explore this topic, we will modify a Terminal program that will, when complete:

- Open a file and write a number of records to the file. Close the file.
- Open a file and loop according to the number of records to read in each record.
- Print each record that is read (as they are read).

Use the code provided to get started, using this code complete the following:

1. Open and look at the code provided for reading and writing the records from files. The functionality of this code is basically correct, but the code can be improved in design and implementation, these are the modifications you will make.
2. Make the following modifications the **basic_read_write.rb** program so that it:
 - Uses a loop in `read_data_from_file()`, with the loop controlled by the number at the start of the file.
 - Improve the functional decomposition by removing as many lines of code from main as possible, yet retaining good structure.



NB: you must change `read_data_from_file()` and `write_data_to_file()` so that they take a text filename rather than a file descriptor.

The program output should look as follows:

```
MacBook-Pro-6:4.1T File Handling mmitchell$ ruby basic_read_write_answer.rb
Fred
Sam
Jill
Jenny
Zorro
MacBook-Pro-6:4.1T File Handling mmitchell$
```



Hint: you need to change what you are given so as to have the following procedures:
`read_data_from_file(file_name)` and `write_data_to_file(file_name)` - note the parameter change from file to filename.

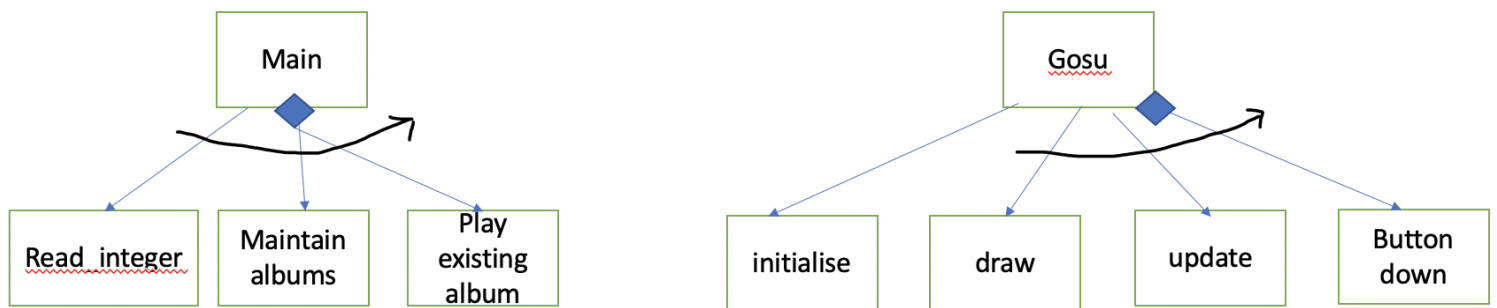
4.2T M □ Gosu Major Cycle

Modify a simple Ruby program to move a shape across the screen by modifying the tasks in the Gosu cycle `update()` and `draw()` methods.

Enhance the code provided as follows:

- Add a variable in the `initialize()` method called `shape_x` with the initial value of zero.
- Add code into the `update()` method that will add 10 to `shape_x`.
- Add code into the `draw()` method that will draw a shape (square or circle of any visible colour) at the y coordinate of 30 and x coordinate of `shape_x`.

Regular main() compared with GOSU cycle



Use the code provided to get started. You also need to download the **media** folder with its contents.

Upload a screenshot of your code running to the workspace.

4.3C M □ Gosu Shape Moving

Modify a simple Ruby program to move a shape across the screen by changing the provided code in the Gosu cycle `update()` method.

Use the code provided to get started.

You must enhance the code provided as follows:

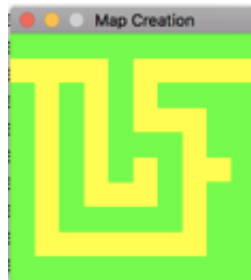
1. The shape also can be moved up and down
2. The shape does not move out of the window area

Upload a screenshot of your program running to the workspace.

4.4D M □ Maze Creation

You must complete the code so that it works as follows:

1. The code in the `initialize()` procedure of the `Gosu` window should be completed to set up cells that are connected to each other with variables joining each cell to its neighbours (using references).
2. The user should be able to left click on cells on the screen to create mazes (and later in the Maze Search task we will use recursion to find a path through the maze).
3. Each cell clicked on should turn yellow.



4. Once this is working (or perhaps before) add code to print out each cell and indicate whether the reference to the neighbour on each side is nil or not, as per the following:

```
Cell x: 0, y: 0 north:0 south: 1 east: 1 west: 0
```

In this case there is no neighbour to the north or the west.

(NB: whether the east or the west neighbour is nil will depend on your perspective – i.e is your perspective looking into the screen, or out of the screen)

Your submitted screenshot should look something like the following:

```
MacBook-Pro-6:4.4 HD Maze Creation (ADD PRINT REQUIREMENT) mitchell$ ruby gosu-maze-creation-answer.rb
Cell x: 0, y: 0 north:0 south: 1 east: 1 west: 0
Cell x: 0, y: 1 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 2 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 3 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 4 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 5 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 6 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 7 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 8 north:1 south: 1 east: 1 west: 0
Cell x: 0, y: 9 north:1 south: 0 east: 1 west: 0
----- End of Column -----
Cell x: 1, y: 0 north:0 south: 1 east: 1 west: 1
Cell x: 1, y: 1 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 2 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 3 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 4 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 5 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 6 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 7 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 8 north:1 south: 1 east: 1 west: 1
Cell x: 1, y: 9 north:1 south: 0 east: 1 west: 1
----- End of Column -----
Cell x: 2, y: 0 north:0 south: 1 east: 1 west: 1
Cell x: 2, y: 1 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 2 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 3 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 4 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 5 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 6 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 7 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 8 north:1 south: 1 east: 1 west: 1
Cell x: 2, y: 9 north:1 south: 0 east: 1 west: 1
----- End of Column -----
Cell x: 3, y: 0 north:0 south: 1 east: 1 west: 1
Cell x: 3, y: 1 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 2 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 3 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 4 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 5 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 6 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 7 north:1 south: 1 east: 1 west: 1
Cell x: 3, y: 8 north:1 south: 1 east: 1 west: 1
```



i Once you have the program working as required submit a screenshot to the workspace.

Sobkowicz, M 2015 *Learn game programming with Ruby : bring your ideas to life with Gosu*, The Pragmatic Bookshelf (See chapter 7 for help the grid aspect of the Maze Task)

[Gosu Ruby Documentation](#)

[Gosu site](#)

[Gosu game video tutorial](#)

[Sobkowicz, M 2015 *Learn game programming with Ruby : bring your ideas to life with Gosu*, The Pragmatic Bookshelf \(See chapter 7 for help the grid aspect of the Maze Task\)](#)

[Gosu Ruby Documentation](#)

[Gosu site](#)

[Gosu game video tutorial](#)

5.1T □ Track File Handling

Use the code provided to get started.

You must enhance the code provided as follows:

- Complete the provided code so that it reads in an array of tracks then prints them out.

The program output (using the file input.txt) should look as follows:

```
MacBook-Pro-6:TuteTasks mmitchell$ ruby music_tracks_only.rb
Crackling Rose
sounds/01-Cracklin-rose.wav
Soolaimon
sounds/06-Soolaimon.wav
Sweet Caroline
sounds/20-Sweet_Caroline.wav
MacBook-Pro-6:TuteTasks mmitchell$ █
```

Once your program is running upload a screenshot to the workspace.

Below is a structure chart for this task:



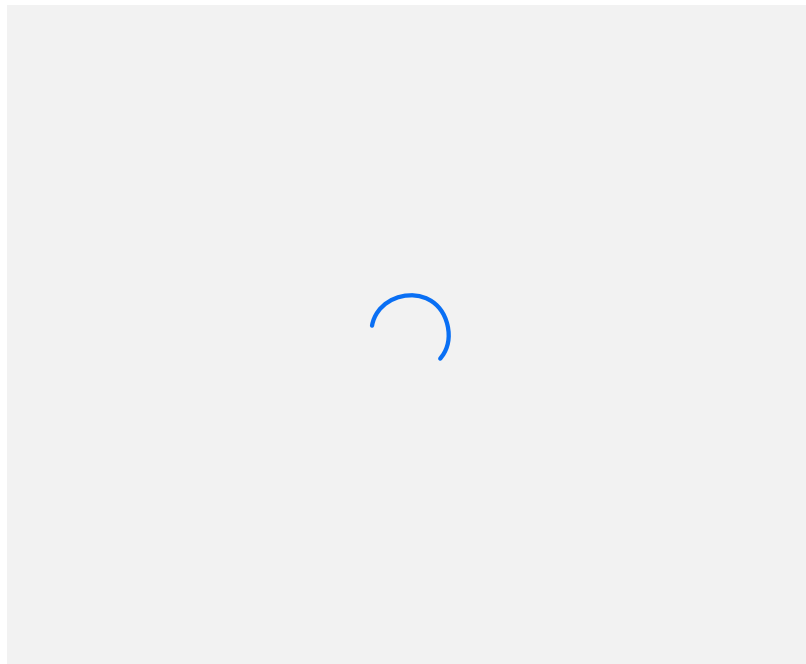
5.2T □ Music Records

Use the code provided to get started.

You must enhance the code provided as follows:

1. Add the missing code to the function/method `read_album()`
2. Add the missing code to the procedure/method `print_album()`
3. Optional: Add an `initialize()` method to the Album class/record definition.

The code should work similar to the following when run:



Once your program is running upload a screenshot to the workspace.

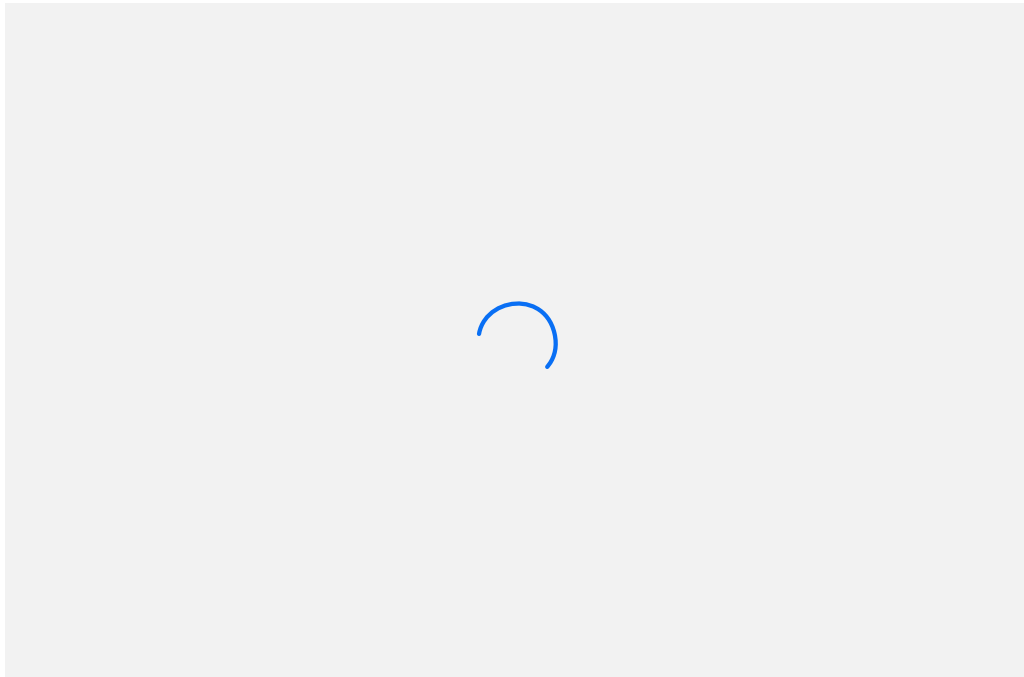
5.3C M □ Hover Button

Make the following corrections to the program `hover_button_test.rb`:

- The button should have a black border around it when the mouse is moved over it to highlight it.
- At the bottom of the screen there should be a display of the mouse x and y locations at all times (not just when the mouse is clicked)
- When the button is clicked the background should change to yellow, if the window area outside the button is clicked the background should change to white.
- Make sure the button works as per the test data below:



The screen should look as follows when the mouse is over the button:



Once your program is running upload a screenshot to the workspace.

6.1T □ Array Search

In task 5.1 you did the following:

Read in a number of tracks **from a file**.

In THIS task you must: (NB. Build on your code for Task 5.1)

- Once the track information is loaded, prompt the user to enter a search string (use the `read_string()` function from the `input_functions` library).
- The search function should have the following prototype: `search_for_track_name(tracks, search_name)` – where `tracks` is an array of tracks and `search_name` is the string being searched for. The function should return -1 if the track name is not found, otherwise it should return the array index of the track with that name.
- Display a message indicating if the track has been found or not (this is provided in the sample code in Resources)
- Make sure you test for a track that does exist and one that does not.

An example of the expected output of your program is provided below:



6.2 P □ Album File Handling

The program must read in a **single** album and a number of tracks for the album as well as a genre for the album **from a file**. You can have as many genres as you like, but these must be defined using an enumeration. **For each track you must read in a track name and a track filename so re-use the code from task 5.1 T** (also see 5.2T for the album/music record).

Your application must read the album and track information **from the provided album.txt** file. Use the provided code `album_file_handling.rb` as a basis for your program.

The output of your program should look as follows:

Once your program is running upload a screenshot to the workspace.



6.3 D M □□ Custom Program Design

In this task you will provide a plan and overview of the structure of a custom program (something you would be interested in creating).



[Credit Task 6.3 - Custom Program Design.docx](#)

Specifically it should:

1. Demonstrate the use of functional decomposition - implement the program with a number of functions and procedures. (Maybe even modular decomposition with separate units if you can identify some reusable artefacts - optional but nice)
2. Demonstrate the use of arrays and records
3. Demonstrate the use of structured programming (sequence, selection, and repetition)

Here are some steps to get you started:

1. Download the Design Report template attached above.
2. Provide a summary of your program — What does it do? What are some of the key features etc.
3. Describe the main data types: records and enumerations.
4. Describe the main functions and procedures. Get some detail down now for your tutor to check, but there is no need to spend ages on this task. Have enough that you can start to see how the program will continue to develop as you proceed.
5. Show your plans to your tutor, lecturer, help desk staffers, and/or friends to get some feedback.

Note: Your program should be different from the food hunter program and the lecture demonstration programs. You want to demonstrate that you have learnt from these tasks and can apply what you have learnt to some other program design.

If you are aiming for a High Distinction, review the related High Distinction Project document for details on how you can ensure this program meets the HD requirements.



Note: Your program should be different from the food hunter program and the lecture demonstration programs. You want to demonstrate that you have learnt from these tasks and can apply what you have learnt to some other program design. If you are aiming for a High Distinction, review the related rubric (attached below) for details on how you can ensure this program meets the HD requirements.



[MarkingRubricSummary\(4\)-1.pdf](#)

Once your report is finished upload it to the workspace.

Week 7 Check Point: □ Feedback on Tasks up to Week 6

By the end of Week 7 you should have received feedback on the manually marked tasks up to and including 5.4 CT.

If you needed to REDO any of these tasks, you should look at the tutors feedback in the task submission and make any corrections.

Once your corrections are complete you MUST notify your tutor and ask that the task be re-marked.

Demonstration of the Text Music Player (7.1)

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

7.1P M □ Text Music Player



Do NOT load your song files to Ed.

In this task you will extend the implementation of your Text Based Music Player.



The grade achievable for this task (within the Pass range) is:

- **Basic Pass Level - 55 (if you complete all Tutorial and Pass tasks to required standards)**

You can see a demonstration of how this program works at: [Text Music Player Demonstration](#) (acknowledgement to Mathew Wakefield) see below:

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

To see how to create the file with the albums (and design the code) see the following:

<https://echo360.org.au/media/befb248b-a61b-4965-bb37-cc717e8906d1/public>

Resources:

- Frieder, O. Frieder, G. & Grossman, D. 2013 Computer Science Programming Basics in Ruby, O'Reilly Media (Chapter 6)
- Flanagan, D. & Matsumoto, Y. 2008 The Ruby Programming Language, O'Reilly.
- Pine, C 2014, *Learn to Program (2nd Ed), Chapter 11*, The Pragmatic Programmer (library version – follow the link)

Pass Level Requirements

Your Text Based Music Application must have the following functionality:

Display a menu that offers the user the following options:

1. Read in Albums
2. Display Albums
3. Select an Album to play
4. Update an existing Album
5. Exit the application

Menu option 1 should prompt the user to enter a filename of a file that contains the following information:

- The number of albums
- The first album name
- The first artist name
- The genre of the album
- The number of tracks (up to a maximum of 15)
- The name and file location (path) of each track.
- The album information for the remaining albums.

Menu option 2 should allow the user to either display all albums or all albums for a particular genre. The albums should be listed with a unique album number which can be used in Option 3 to select an album to play. The album number should serve the role of a 'primary key' for locating an album. But it is allocated internally by your program, not by the user.

Menu option 3 should prompt the user to enter the primary key (or album number) for an album as listed using Menu option 2. If the album is found the program should list all the tracks for the album, along with track numbers. The user should then be prompted to enter a track number. If the track number exists, then the system should display the message "Playing track " then the track name, " from album " then the album name. You may or may not call an external program to play the track, but if not the system should delay for several seconds before returning to the main menu.

Menu option 4 should allow the user to enter a unique album number and change its title or genre. The updated album should then be displayed to the user and the user prompted to press enter to return to the main menu (you **do not** need to update the file).



NB: IF you are aiming for a CREDIT or higher in the unit, then you should move on to doing your GUI Music Player (CT7.2).



Submit your final code and a screenshot to the workspace.

7.2C M □ GUI Music Player



Do NOT load your song files to Ed. Your tutor will ask you to demonstrate your code in class.

In this task you will build on the skills developed in your other Pass, Credit and Tutorial tasks.

There is a minimum requirement for naming and design before your code can be accepted – regardless of how well the code is functioning.



The grade achievable for this task (within the Credit range) is:

- Credit Level – 65 (if all tutorial, pass, and credit tasks are completed to required standards)



Note: See the Gosu audio API at <https://www.rubydoc.info/github/gosu/gosu>. You will need to use the following: `Gosu::Song.new(filelocations)` – create a new song; and: `song.play(false)` – play the song.

Credit Level Requirements - 65



Before doing this level complete the task CT7.1

The program must read in (from a file) a single album and up to 15 tracks for the album.

The information read from the file should include:

- Album title
- Artist
- Artwork file name (place your artwork in an /images folder under the main folder)
- The number of tracks
- The title of each track
- The file location of each track (again, use a sub-directory/folder to store the song files.)

At this level user interaction must be entirely through a GUI. Your GUI interface should show a single album using either a text description, artwork or both. Users should be able to click on the Album information (i.e the artwork) and the tracks will be listed and the first track start playing. The tracks should continue to play in order until either they are finished or the program is stopped. The currently playing track must be indicated somehow (e.g the track could be highlighted or display a simple text message 'Now playing ..'). You must use the Gosu audio API for this component.

Your GUI may look something like the following:



i Submit your final code and a screenshot to the workspace.

7.3D M □ Extended GUI Music Player



Do NOT load your song files to Ed. Your tutor will ask you to demonstrate your code in class.

In this task you will build on the skills developed in your other Pass, Credit and Tutorial tasks.

There is a minimum requirement for naming and design before your code can be accepted – regardless of how well the code is functioning.



The grade achievable for this task (within the Distinction range) are:

- **Distinction Level – 75 - This task.**



Note: See the Gosu audio API at <https://www.rubydoc.info/github/gosu/gosu>. You will need to use the following: `Gosu::Song.new(filelocations)` – create a new song; and: `song.play(false)` – play the song.

Distinction Level Requirements - 75



Before doing this level complete the tasks CT7.1 and CT7.2

At this level your program must read in (from a file) at least four albums and up to 15 tracks for each album.

The information read from the file should include:

- Number of Albums
- Album title
- Artist
- Artwork file name (place your artwork in an /images folder under the main folder where you run the program)
- The number of tracks
- The title of each track
- The file location of each track

At this level user interaction must be entirely through a GUI. Your GUI interface should show all the albums using either a text description, artwork or both. Users should be able to click on any Album information (i.e the artwork) and the tracks will be listed. **The user should then be able to click on a track to play that track. The currently playing track must be indicated somehow (e.g the track could be highlighted or display a simple text message 'Now playing ..'). If the user clicks on another track (for the current album or another album) then any currently playing track should be stopped and the most recently selected track start playing.**

You must use the `Gosu` audio API for this component.



Submit your completed code to Ed.

Custom Program Extensions (for higher Distinction Grades or for High Distinction Custom Program)

You may wish to extend on the requirements above for a higher distinction grade or for your custom program. You would need to discuss this with your tutor as to what is required for different grade levels. Some possible extensions are:

Possible custom program Distinction level Extensions:

- Allow users to page through multiple pages of albums.
- Allow sorting of albums based on year recorded, genre ,etc.

Possible Custom program High Distinction extensions:

- Allow users to select tracks from various albums to create playlists. Users can then select from the playlists to play a pre-selected sequence of tracks.
- Some combination of all the above extensions.



Submit your final code and a screenshot to the workspace.

Test 1 □ □ Week 8 (2pm Wed or 7 pm Thurs)

In Week 8 you will sit a timed test in the workshop time.

This test will be available in a lesson (NOT in these portfolio tasks).

Your tutor will mark your test and give you feedback here on whether you have:


- 1) Passed
- 2) Need to fix and resubmit the test
- 3) Need to REDO the test in another sitting.

8.1T M □ □ Concept Map

In your tutorial/lab groups produce a concept map in preparation for the test next week. You should cover the concepts from the lessons and tasks. First make a list of terms, then draw a diagram that organises those terms into a comprehensive overview that relates the concepts and terms to each other.

Here are some of the things you should cover:



 Submit your completed concept map to the workspace.

8.2D M ☐☐ Food Hunter

In this task you are provided with the Ruby source code for a version of the Food Hunter program. You are required to extend that program to implement the changes specified below.

 [Resources.zip](#)

The following steps will guide you to complete this task.

- 1.Download the **Ruby Food Hunter** starter code (attached).
- 2.Make the changes as specified below to the provided **foodhunter.rb** code.



(If the video doesn't work [use this link](#))

3. Save your code using the name: `foodhuntermodified.rb`.

4. Answer the Programming Principles questions on the provided Question/Answer sheet (see the Resources attached for this task).



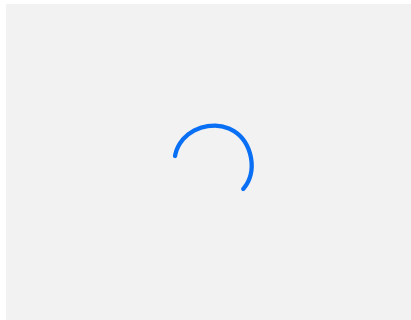
See the following resources to help with this task:

- [Pine, C 2009, 'Learn to Program', Pragmatic Bookshelf \(in the library\)](#)
- [Sobkowicz, M 2015 *Learn Game Programming with Ruby: Bring Your Ideas to Life with Gosu*, The Pragmatic Programmer.](#)
- [Tutorial for the gosu game package.](#)
- [Set of videos for Ruby.](#)

Submit your final answers to the workspace as follows:

1. Your **foodhuntermodified.rb** code.

2. Your screenshot of your code running. It should look something like the following:



3. Your answers to the question sheet.



Submit your code and a screen shot to the workspace and Doubtfire.



See the hint code in the file attached here. Draw will need to use the `changing` variable for food.



[Screen Shot 2021-05-05 at 10.09.30 am.png](#)

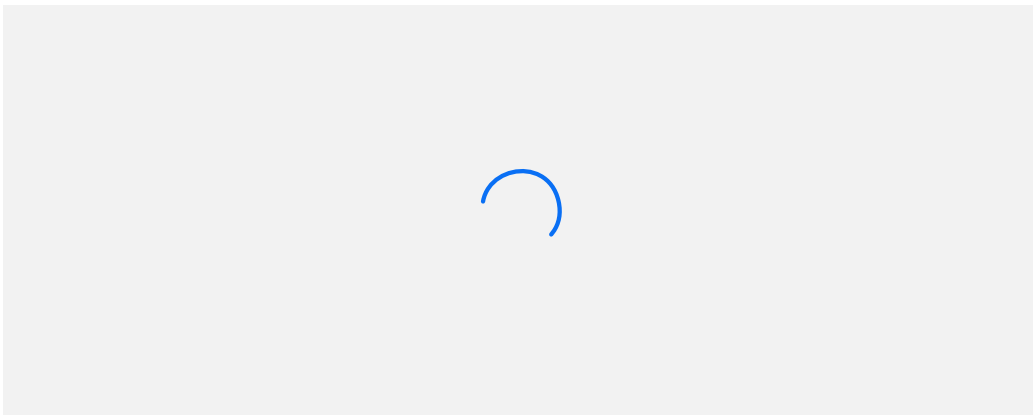
9.1T M □ Fix it

In this task you will build on the design and debugging skills developed in your other Pass and Tutorial tasks.

You will work together in a tutorial to:

1. Get the program working as it is.
2. Redesign the code to improve the modularity, coupling and cohesion.
3. Test and demonstrate your redesigned code.

Below is the expected output of the fixed program:



Flanagan, D. & Matsumoto, Y. 2008 The Ruby Programming Language, O'Reilly.

Pine, C 2014, *Learn to Program (2nd Ed)*, Chapter 11, The Pragmatic Programmer (library version – follow the link)



Resources:

Flanagan, D. & Matsumoto, Y. 2008 The Ruby Programming Language, O'Reilly.

Pine, C 2014, *Learn to Program (2nd Ed)*, Chapter 11, The Pragmatic Programmer (library version – follow the link)



Submit your code and a screenshot to Doubtfire.

9.2D □ Custom Code



Note: You have a choice to do the Distinction level Extended Music Player or another Custom Program - approved by your tutor. This explains the Custom Program option.

You are now close to completing tasks related to all of the unit learning outcomes, and can work toward demonstrating these in your own program. If you are aiming for a Distinction or higher grade you should start working on this program now. Aim to create something of at least the complexity of the original Food Hunter program for the lower distinction grade or more complex for higher grades. Specifically it should:



[MarkingRubricSummary\(4\)-1.pdf](#)

1. Demonstrate the use of functional decomposition - implement the program with a number of functions and procedures. (Maybe even modular decomposition with separate units if you can identify some reusable artefacts - optional but nice)
2. Demonstrate the use of arrays and records
3. Demonstrate the use of structured programming (sequence, selection, and repetition)
4. Demonstrate appropriate use coding conventions - case, indentation
5. It must not use global variables, or goto.
6. Make sure you can explain your code in an interview!
7. Use the checklist on the next page to make sure you have everything you need to submit!



Here are some steps to get you started:

1. Think about what you want the program to do. Maybe write up a paragraph or two to explain it to others. Drawing a picture of what you want it to look like is also a great idea.
2. Show your plans to your tutor, lecturer, help desk staffers, and/or friends to get some feedback.
3. Start thinking about the data - what records and enumerations will you need?



Submit your code and a screenshot of it to the workspace.

9.3HD □ Custom Code

You are now close to completing tasks related to all of the unit learning outcomes, and can work toward demonstrating these in your own program. If you are aiming for a Distinction or higher grade you should start working on this program now. Aim to create something of at least the complexity of the original Food Hunter program for the lower distinction grade or more complex for higher grades. Specifically it should:



[MarkingRubricSummary\(4\)-1.pdf](#)

1. Demonstrate the use of functional decomposition - implement the program with a number of functions and procedures. (Maybe even modular decomposition with separate units if you can identify some reusable artefacts - optional but nice)
2. Demonstrate the use of arrays and records
3. Demonstrate the use of structured programming (sequence, selection, and repetition)
4. Demonstrate appropriate use coding conventions - case, indentation
5. It must not use global variables, or goto.
6. Make sure you can explain your code in an interview!
7. Use the checklist on the next page to make sure you have everything you need to submit!



Here are some steps to get you started:

1. Think about what you want the program to do. Maybe write up a paragraph or two to explain it to others. Drawing a picture of what you want it to look like is also a great idea.
2. Show your plans to your tutor, lecturer, help desk staffers, and/or friends to get some feedback.
3. Start thinking about the data - what records and enumerations will you need?



i Submit your code and a screenshot of it to the workspace.

Week 10 Checkpoint: □ Feedback on tasks up to Week 9

By the end of Week 9 you should have received feedback on the manually marked tasks up to and including 8.3 DT.

If you needed to REDO any of these tasks, you should look at the tutors feedback in the task submission and make any corrections.

Once your corrections are complete you MUST notify your tutor and ask that the task be re-marked.

Test 2 □□ (2 pm Wed or 7pm thurs)

In Week 10 you will sit a timed test in the workshop time.

This test will be available in a lesson (NOT in these portfolio tasks).

Your tutor will mark your test and give you feedback here on whether you have:

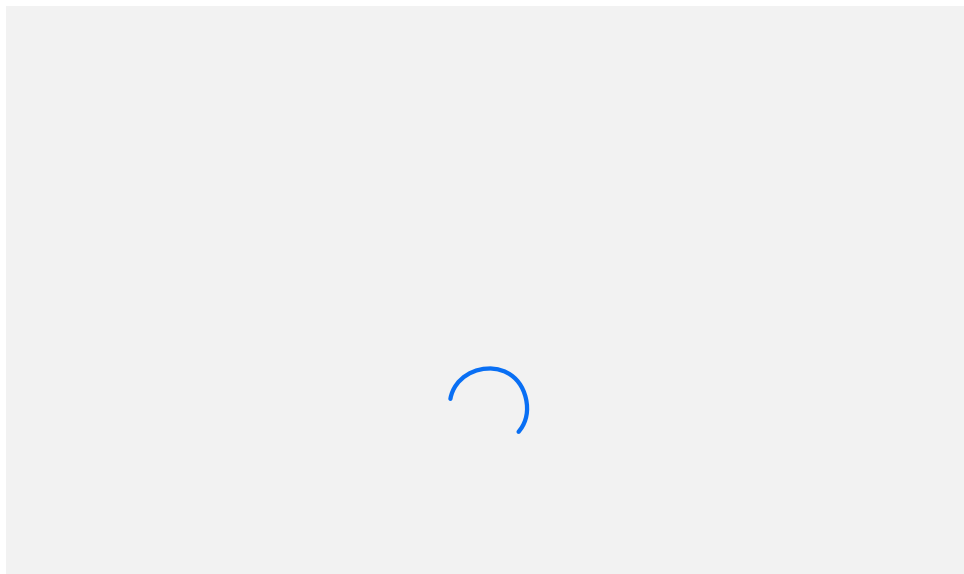
- 1) Passed
- 2) Need to fix and resubmit the test
- 3) Need to REDO the test in another sitting.

10.1.1P □ 'Hello World' in C

In this task you will compile and run a “Hello World” terminal program in C.

To compile your program you will need to use **mingw** (in Windows) or the **Terminal** (Mac OS):

1. If you don't already have one, make a directory (i.e., a 'folder') to store your code (e.g., *Documents/Code/Lab1*). On a Swinburne computer you may wish to use a directory on your student drive or a USB storage device.
 1. • Navigate to your *Documents* directory in Finder or File Explorer
 2. • Right click in the *Documents* directory and select **New Folder**, name it **Code**



2. Open a **Terminal** (MSYS or MinGW shell in Windows), then perform the following commands:

- Change into the directory containing your code using the **cd** command.

```
cd /c/Users/your_user/Documents/Code on Windows or
```

```
cd ~/Documents/Code on MacOS or Linux
```

-
- List the files in this directory using the **ls** command
 - Print the working directory using the **pwd** command
 - Compile your program using **gcc hello_world.c**
 - List the files in this directory using the **ls** command to see the files created by the compile process

- Run your program using **`./a.out`**



Use the sample code provided in the Resources for this task and modify that code to meet the requirements as described above.

10.1.2P □ 'Hello World' in Python

Write and run a program to print "Hello World!" using Python.



Hint: the code looks exactly the same as if you were printing a string in Ruby without a newline.

■ Compile and run your Python program:

Run and compile your python program by typing: **python3 hello_world.py**

Reference: Speight, A 2020 *Bite Sized Python*. Wiley. [Available online from Swinburne Library.](#)

See also: <https://www.python.org/dev/peps/pep-0008/#prescriptive-naming-conventions>

10.2C □ Recursive Factorial

Building on the provided code in this task's resources, write a recursive function that calculates a factorial using recursion.

The program will take a number on the command line and calculate the factorial for that number.

You also need to put in a check in `main()` case an incorrect argument is passed in. You will need to include the following line of code:

```
puts("Incorrect argument - need a single argument with a value of 0 or more.\n")
```

You need also to add error checking into main. . The output should look as follows:



Include your tests in your screen shot which you submit to Doubtfire

10.3HD M □□ Maze Search



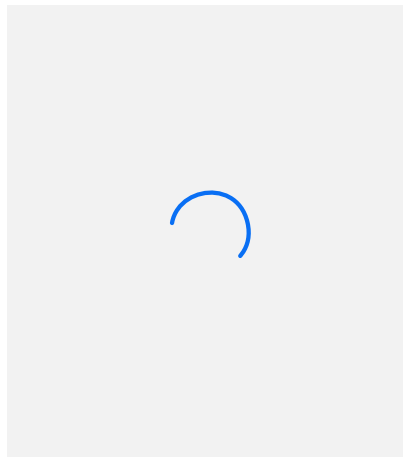
NOTE: You cannot run Graphical programs in this tool (ED) - you need to copy the provided code to your IDE and run it on a computer with GOSU installed.

Use the code you completed in Task 4.4D to get started.

You must complete the code so that it works as follows:

1. Once the user has used the Left Mouse Key to create at least one path from one side of the screen to the other, the user should then be able to Right Click the mouse on a yellow cell on the left most side of the screen.
2. The program should then search using the recursive *search()* function to find a path to the right side of the screen following one of the user's created paths.
3. Each cell on the path should then be displayed in red.

A found path through the maze may look something like:





Once you have the program working as required submit your code and a screenshot to the workspace.

Sobkowicz, M 2015 *Learn game programming with Ruby : bring your ideas to life with Gosu*, The Pragmatic Bookshelf (See chapter 7 for help the grid aspect of the Maze Task)

[Gosu Ruby Documentation](#)

[Gosu site](#)

[Gosu game video tutorial](#)

10.4HD M □ □ Custom Project

The aim of this task is to demonstrate significant depth of understanding related to the unit's topics and concepts. Several ideas are presented, but you are free to do this in any way you see fit.



[MarkingRubricSummary\(4\)-1.pdf](#)

This task works in conjuncture with the HD standards on the Custom Program. With the custom program you are demonstrating that you can apply the concepts learnt, whereas in this task you are demonstrating that you can explain and discuss the concepts with similar depth of understanding.

There are a number of things you can do to help demonstrate your ability to explain and discuss programming concepts at depth. The following types of suggestions indicate appropriate projects in ascending order of grade level for this task:

- **Basic (90 - 93):** Provide a tutorial on the use of Gosu, Tcl/Tk or other Ruby gems to accomplish a task.
 - This could be a walkthrough to write a game, or use some more advanced features (networking, web, sprites, physics, etc).
 - For higher grade levels you should relate use of the libraries/gems to coding with good program design and abstraction.
 - You must have primary source references.
- **Intermediate (93-95):** Explain a concept. The concept must be challenging, and you should do BOTH the following:
 - Write a research report on the concept explaining it and drawing on the literature.
 - Suitable concepts could be programming pattern or design principle.
 - Produce a video. Explain your concept. Maybe show example code to help with this.
 - You must have primary source references.
- **Advanced (95+):** Conduct a small research project aiming to answer a question related to programming or program design, or propose a new design pattern or practice.
 - Create a plan to outline the question and method for your research project. The **research question** is the question you aim to investigate in the project. The research method describes how you will approach answering the question.
 - Collect evidence to address your research question.
 - You must have primary source references.

Submit your project files to the workspace.

Week 11 Checkpoint: □ Feedback on tasks up to Week 10 (except custom code)

By the end of Week 10 you should have received feedback on the manually marked tasks up to and including 10.4 HDT. **(But NOT the custom code tasks - those are assessed in the interviews after Week 12)**

If you needed to REDO any of these tasks, you should look at the tutors feedback in the task submission and make any corrections.

Once your corrections are complete you MUST notify your tutor and ask that the task be re-marked.



NB: the remaining manually marked tasks (for Week 11 and Week 12) will be assessed when portfolios are assessed in the exam period. But you can seek feedback from tutors in tutorials and help desk sessions.

11.1T M □ □ Learning Summary

The Learning Summary Report is your chance to outline how the work you have completed demonstrates that you have met all of the unit's learning outcomes. In this document you will indicate the grade you are applying for, and provide reasons why you should be awarded this grade based on the unit's assessment criteria.



[COS00002-COS10009-COS60006 Portfolio Process and Assessment Criteria.pdf](#)

This task will be assessed when we mark your portfolio after week 12.

Complete the following form and upload as a pdf file.



[LearningSummaryReportTemplate.docx](#)

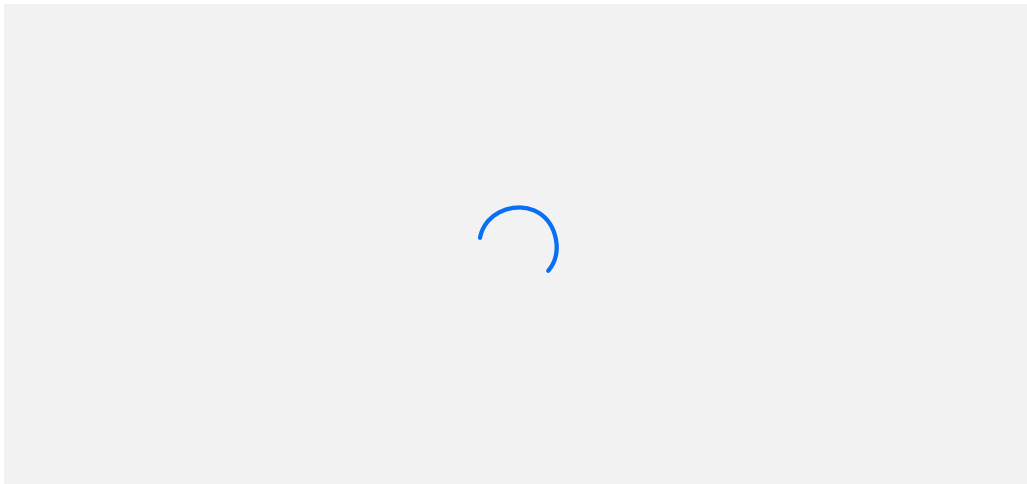
11.4D □ C Name Tester

In this task you will redo your Silly Name program from Week 3 and write it in the C programming language.

Your program should:

- Have a *main()* which prompts the user for a name
- But **you need to move some code out of `main()` into a procedure (void function)** that checks the name and prints the appropriate message.

The pseudocode for this procedure follows:



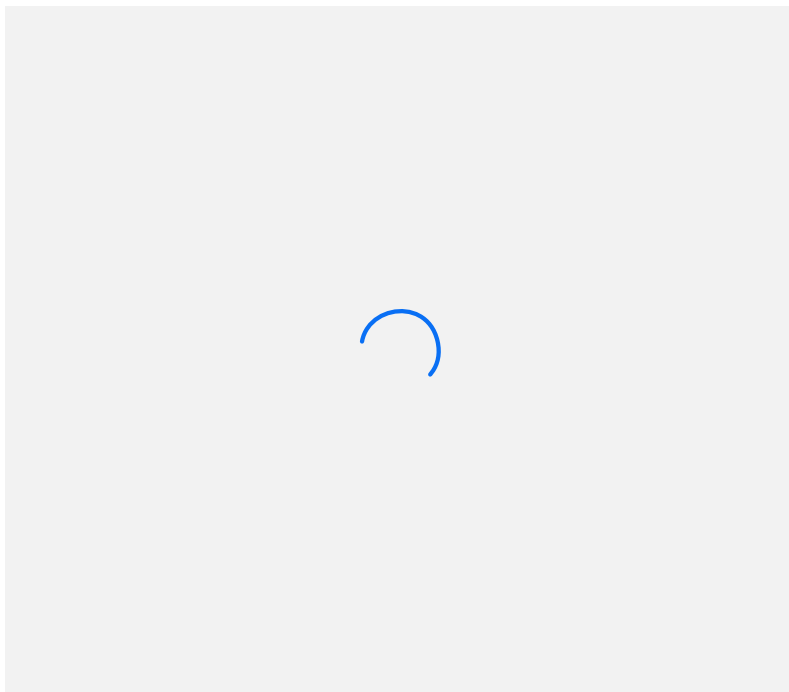
Compile your program using the following Terminal command:



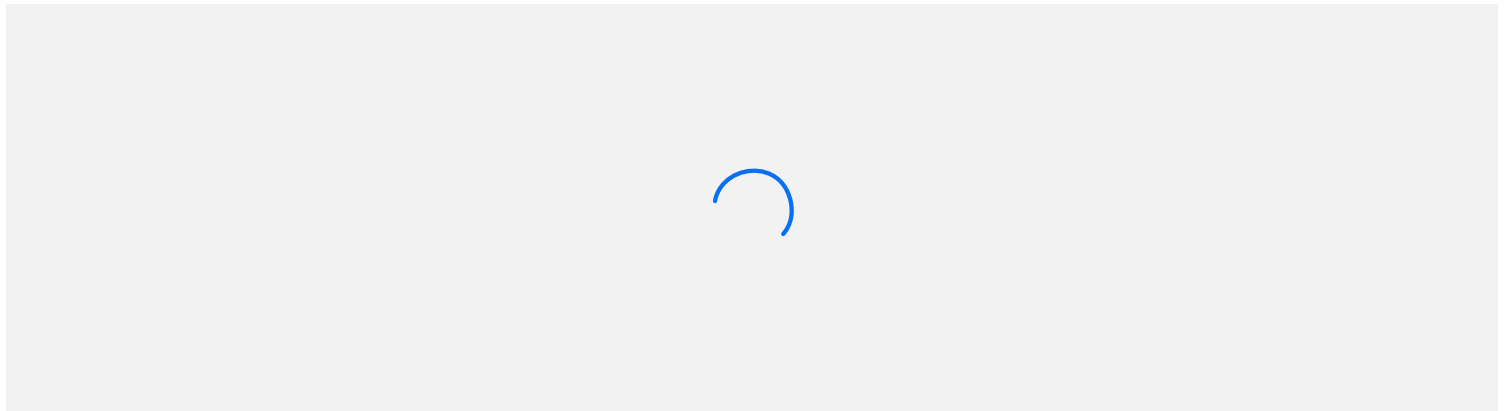
run your code by typing: `./a.out`


You can compile and run the code as is, then make your changes and check it still works.

Now `main()` should be CHANGED TO LOOK AS FOLLOWS:



Your output should look as follows:



 Submit your code and a screen shot of it running to Doubtfire.

11.5D/HD □□ Custom Code Video

Just upload here a document with a link to your custom code video.

11.2P □ Python Program

In this task you will redo your Silly Name program from Week 3 and write it in the Python programming language.

Modify the provided code to produce the following output:



11.3C □Python Shape Moving

You must enhance the code provided as follows:

- 1.The shape also can be moved up and down
- 2.The shape does not move out of the window area.



You will need to install python3 and pygame on your machine for this task. See below taken from:
<https://www.pygame.org/wiki/GettingStarted>

Pygame Installation

Pygame requires Python; if you don't already have it, you can download it from python.org. **Use python 3.7.7** or greater, because it is much friendlier to newbies, and additionally runs faster.

The best way to install pygame is with the [pip](#) tool (which is what python uses to install packages). Note, this comes with python in recent versions. We use the --user flag to tell it to install into the home directory, rather than globally.

```
python3 -m pip install -U pygame --user
```

To see if it works, run one of the included examples:

```
python3 -m pygame.examples.aliens
```

If it works, you are ready to go! If not there are more detailed, platform-specific instructions [here](#).

□ Portfolio Completion Quiz

Question

Select which of the following are true in relation to completing the requirements for different grade levels:

- ☐ All pass and tutorial tasks must be completed along with both tests, in order for a student to pass the unit.
- ☐ All the tasks for higher levels must be completed to achieve that level.
- ☐ For a distinction above 70 or high distinction a custom program is required.
- ☐ It is ok just to attempt tasks, they do not actually need to be completed.
- ☐ Portfolio submitted in Doubtfire

Week 12 Checkpoint: □ Feedback on Tasks up to Week 11

By the end of Week 11 you should have received feedback on the manually marked tasks up to and including 10.3 CT.

If you needed to REDO any of these tasks, you should look at the tutors feedback in the task submission and make any corrections.

Once your corrections are complete you MUST notify your tutor and ask that the task be re-marked.

12.1C □ Minitest Test Harness

Using the Ruby `minitest` gem, write and run tests for the code provided for this task.

Complete the tests required (as indicted by the comments) in `tester_demo.rb`.



Submit your code and a screenshot - **NOTE: can be completed after WEEK 12 (i.e does not need to be marked complete)**

Week 13 Checkpoint: ☐ All tasks submitted

Well done! If you have completed all the tasks, then the following will be looked at when your portfolio is marked during the examination period:

- 11.2 Learning Summary
- 9.2 and 9.3 Custom Project (if you are aiming for a HD)
- 11.5 Custom Project (if you are aiming for HD band 2)