Open Awards

Quality Endorsed Unit

1 Unit Details

Unit Title:	Ruby Programming For Beginners
Unit Reference Number:	CK3/3/WR/013
Level:	Level 3
Credit Value:	8
Minimum GLH:	45

2 Learning Outcomes and Criteria

Learning Outcome (The Learner will):	Assessment Criterion (The Learner can):
1. Understand how to install the Ruby interpreter and run Ruby programs.	1.1 Complete a Ruby installation under the Microsoft Windows operating system.
	1.2 Use IRB (Interactive Ruby) to input Ruby commands and statements.
	1.3 Create a simple Ruby program and run it in both IRB and in a command prompt window.
2. Understand Ruby variables and data types.	2.1 Illustrate the use of variables, their naming conventions, and scope in a Ruby program.
	2.2 Incorporate Ruby strings and a range of string methods into a Ruby program.
	2.3 Perform basic arithmetic using a range of Ruby's arithmetic operators.
	2.4 Deploy Ruby's comparison and logical operators in a Ruby program.
	2.5 Manipulate and concatenate Ruby strings and work with a range of numeric data types.
3. Understand Ruby's branching and looping statements.	3.1 Incorporate the IF, ELSE, and ELSIF statements into a Ruby program.
	3.2 Insert FOR, UNTIL and WHILE loops into a Ruby program.
	3.3 Create a "Guess the Number" game program combining Ruby's branching and looping statements.

4.	Understand Ruby Arrays and Hashes.	4.1	Create a “Hangman” game and: a) Store words in an array; b) Generate a random word from the array and display _ in place of each letter in an answer variable; c) Ask the user to guess a letter; d) Check the guess is correct; e) If guess is correct, insert the letter in the correct place or places in the answer variable, else decrement the number of attempts; f) Check for end of game, which is where the user has either guessed the word or has run out of guesses.
		4.2	Create a Ruby array and: a) Add an item to the array; b) Insert an item between existing items; c) Delete an array item; d) Add an additional array; e) Organise the arrays into ascending and descending orders; f) Create nested arrays.
		4.3	Create a Ruby hash to: a) Contain a range of key-value pairs; b) Print using a FOR loop; c) Add a new entry; d) Modify an existing entry; e) Delete an entry.
5.	Understand Ruby methods.	5.1	Create a Ruby method, passing arguments to the method and returning values from it.
		5.2	Create an area calculation program to: a) Ask the user to enter width and height; b) Calculate the area using a method; c) Return the calculated area to the main program;
		5.3	Using Ruby methods for each of the following, create a Noughts & Crosses game (where the user plays against the computer) to: a) Display the game’s grid; b) Display instructions; c) Set the difficulty level; d) Randomly decide whether computer or player goes first; e) Get a grid location from player; f) Generate the computer’s move; g) Check for valid moves; h) Check for a winner; i) Ask if the player wants to play again.
6.	Understand how to use Ruby to create and access files and directories.	6.1	Create a directory and a file and: a) Add records to the file; b) Amend existing records; c) Close the file; d) Use a range of directory file methods.
		6.2	Read both characters and lines from a Ruby text file.
		6.3	Create a Ruby Sports Quiz storing the questions and answers in a data file:

			a) Create the data file; b) Open the file in Read Only mode; c) Read the question “blocks” and answer “lines”; d) Check if the player’s answer is correct; e) Keep a score of correctly answered questions; f) After all question “blocks” have been read, display the player’s score and ask if he or she wants to play again.
		6.4	Recognise the various exception types, trap errors using Rescue and Retry, and raise exceptions.
7.	Understand Ruby objects.	7.1	Create a class with multiple methods and instantiate multiple objects from the class.
		7.2	Create an initialize method with attributes.
		7.3	Create a class with private attributes and methods, create multiple properties, and access the properties from outside of the class.
		7.4	Create a Television object as follows: a) Include properties for model, screen size, volume, and channel; b) Provide methods to change the channel and volume; c) Change the object’s properties using the above methods; d) Check if values supplied for volume and channel are within the accepted range.
8.	Understand Object-Oriented Programming techniques.	8.1	Build a multi-class “Highest Card” game, which includes interacting Game, Deck, Hand, and Card classes, with multiple methods and functions.
		8.2	Incorporate inheritance techniques into the “Highest Card” game, changing inherited methods, including polymorphic behaviour, and creating modules.
		8.3	Using object-oriented programming, build a Rock, Paper, Scissors game (where the user plays against the computer), including the following methods: a) Display a greeting; b) Display instructions; c) Play the game; d) Get the player’s “move”; e) Get the computer’s “move”; f) Analyse the results; g) Display the results.
9.	Understand Ruby Graphical User Interface (GUI) development and event-driven programming techniques using the Gosu two-dimensional games library.	9.1	Create a GUI window that includes: a) Drawing objects, such as squares and triangles; b) Text; c) Images d) A background image; e) Animations; f) Mouse and keyboard inputs; g) Colours.

		9.2	Create a “Hit The Object” game, where the user tries to click on a moving coloured square, which bounces off the edges of the graphics window. The game includes: a) A moving 50px square with a background colour; b) Detect if the mouse click is within the moving square and, if so, record a “hit” worth 5 points, else a “miss”, where 1 point is deducted; c) Keep a running total of the player’s score displayed in the top-right corner of the graphics window; d) Set a time limit; e) Detect “end of game” and ask player if he or she wants to play again.
10.	Understand how to develop graphical “collision detection” games using the Gosu games library.	10.1	Create a graphics window containing a background image and moving “sprites”, both automatically and via the mouse.
		10.2	Incorporate collision detection and collision handling techniques into a moving sprites graphics window.
		10.3	Develop a two-player Ping Pong game where a “bouncing” ball has to be prevented from reaching the side of the graphics window by intercepting it with a paddle, controlled by the user’s mouse.
		10.4	Develop a Galaxy War multi-sprite game, where the player controls a Star Ship sprite, which can move forward and rotate and fire “bullets” at a continuous wave of enemy space ships which, when hit, explode and disappear from the screen.
11.	Understand how to develop more advanced games using the Gosu games library in conjunction with the Chipmunk physics engine.	11.1	Build an “Escape” game, where the player controls a character, who is trying to escape from a pit by running and jumping between a range of static and moving platforms, while dodging constantly falling boulders.
		11.2	Incorporate the full range of Chipmunk classes, which hold information concerning the graphical window’s “space”, the velocity, position, and mass of each sprite in the space, the sprites’ “shape” properties, which determine how they react when they collide, and the movement of sprites around the window, taking into account properties such as gravitational pull and damping (or resistance).
		11.3	Build a “Twelve” puzzle game, consisting of a grid of 36 squares, 12 red, 12 green, and 12 blue, distributed randomly around the grid. The player has to move the squares around the board, trying to combine as many like-coloured squares together as possible, by dragging them horizontally or vertically in a straight line onto squares of the same colour. When this happens the squares are combined and the object of the game is to end with one square of each colour.