# **CSP2101 Scripting Languages**

# **Assignment 2: Portfolio 1**

**Student:** Zac Smith

**Student number:** 10459495

**Due date:** 10th April before 1700

## **Script description**

### Set Variables

* **Line 4**: declare **num** variable to be type integer and starting value of ‘0’. Used as a counter of how many guesses it took the user to guess correctly. Counts the number of loops.
* **Line 8**: assigns **numb1** to be integer (whole numbers) only. The ^ caret reads beginning of the line until the end of line operator $. And must contain + values of 0 to 9.

### Set Variables and welcome message to the game

* **Line 11**: **acnum** variable assigned to random number generator. **-i** is the input range to be shuffled. **-n** is the number of lines to be output to.
* **Line 15 & 16**: data.txt assigned to **FILE** variable for later use. A test statement to see if **data.txt** has been previously made by the user. If it has then do nothing **:**, if it hasn’t then it will create data.txt in the users working directory.
* **Line 19**: display welcome message to the user. **-e** so can use the escape operators such as **\n** new line and **\t** tab across to centre of console. **92m** is the colour of the title to be displayed.

### Set user input, user assistance message

* **Line 22**: take the user input asking for a name and assign it to variable **name**.
* **Line 25**: display user assistance message briefly explaining the rules as well as including the users name **$name**. \n display on new line

### cREATE WHILE LOOP, THE MAIN FUNCTION OF THE GAME

* **Line 28**: The while loop keeps iterating the inside statements if **guess** & **acnum** are not equal (loops conditions).
* **Line 31**: the loop counter, each time the loop repeats (the user doesn’t guess correctly) **num** increases by 1, to be used in total number of guesses at the end.
* **Line 32**: assign the users guess to the **guess** variable and display what number guess using loop counter **num**.
* **Line 37-38**: if statement tests both variables, the users **guess** and the **numb1**. This tests the users guess to see if the user entered an integer. If not, it will display the following message (line 38). **!** not equal operator. **=~** tests left to right variables.
* **Line 41-42**: first elif statement. To test the users guess against the random number generated by **acnum**. If **guess** is lower than **acnum**, print the following message (line 42). < less than comparison operator. If this statement is correct, the loop will restart and prompt user for another guess.
* **Line 45-46**: second elif statement. To test user input against the random number generated by **acnum**. If **guess** is greater than **acnum**, print the following message (line46). > greater than comparison operator. If this statement is correct, the loop will restart and prompt user for another guess.
* **Lines 48 & 50**: fi tells the interpreter there are no more if statements. done tells the interpreter the while loop is done and continue onto next lines.

### Extra functionality, display and record game results

* **Line 53**: display the results of the game to the user, using their name that was assigned to the variable **name** done earlier on in the script. Display total number of guesses using the **num** variable that is the loop counter variable. \n display on new line
* **Line 56**: display to console the message. \t\t move message to centre of the screen. \n display on a new line.
* **Line 60**: write the number of guesses (**num**) and the player name (**name**) to the txt file created in line 16. tee -a is appending the results to the text file and not overwriting the contents. >/dev/null is writing the output to the “trash” or bit bucket so it won’t be displayed to the screen via the echo command, as it will be displayed differently below.
* **Line 64**: sort will display the given commands to the terminal. **K2** refers to the second white space in the data.txt file. Reading after the Guesses equal sign, resulting in reading only the number. **-n** will sort the results numerically. **| head -3**. The pipe command so that two commands can be executed at the same time, head -3 means that only the top 3 sorted results will be displayed, giving the user the top 3 players with the least amount of guesses.

## Game output

### A basic game

* The above screen shot shows the welcome message as soon as it is executed, followed by a prompt to the user to enter a name. In the 7 guesses it took for the user to guess correctly, it shows the game hints to say higher or lower.
* After the game is complete and the guess is correct, it displays a congratulations message and the total number of guesses it took the user.
* After the game is played the top results are saved and recorded and displayed.

### Error handling and game records

* The above shows an error message in the first guess, the user input **Nine** which isn’t acceptable in this game, so the error message was displayed to console.
* In the records it shows more people have played and the games were recorded, and the top three players are displayed in the console.

### data file for saving records

* The above is the text file that the game records are sent to. As shown, it is not sorted numerically as this is only done via the sort command in the games script. As well as showing more than 3 users have played, but only the top 3 will be displayed in the console.