COMP 1039

Problem Solving and Programming

Programming Assignment 2

Prepared by:
School of Computer and Information Science
The University of South Australia

Eynesbury Modifications:
Michael Ulpen

TABLE OF CONTENTS

INTRODUCTION

This document describes the second assignment for Problem Solving and Programming.

The assignment is intended to provide you with the opportunity to put into practice what you have learned in the course by applying your knowledge and skills to the implementation of a text-based game of **Blackjack** that uses functions, lists, and file input and output.

This assignment is an **individual task** that will require an **individual submission. You will be required to submit your work via Moodle before Friday Week 12, 11:55pm.**

This document is a specification of the required program and its output. Please ask your lecturer if you do not understand any part of this document or the assignment.

SPECIFICATION - BLACKJACK

We will be writing a simple implementation of a card game called **Blackjack**, also known as **21**. Some of the rules have been changed to make this implementation unique. The goal of the game is to draw cards that add up to a value as close as possible to 21 without exceeding it. A player's total hand value is calculated by adding the values of each their cards. Face cards, such as Jacks, Queens and Kings are all worth 10. The Ace is an unusual card because, if the combined value with other cards in hand does not exceed 21, it will be worth 11, otherwise the Ace will be worth 1. Other number cards are worth the number value shown on the card.

In a standard game of 2-player Blackjack the player and the dealer are dealt 2 cards. The player shows both cards face-up. The dealer shows one of their cards face-up and one face-down. The player decides to draw cards by saying, "hit", or stop drawing cards by saying, "stand". They will draw cards trying to get their total hand value as close to 21 as possible without going over. If the total value of cards in their hand exceeds 21, the player "busts", which means they lose. If the player has not "bust" and they decide to "stand", then it is the dealer's turn to draw cards. The dealer has a special rule that, as soon as their hand value is equal to or greater than 17, they must "stand". If the dealer exceeds 21, the dealer will "bust", which means the player wins. It is possible that both the player and dealer end with the same hand value, in which case the game is tied and this is called, "push".

In this implementation, if either player gets 21 in their first hand of 2 cards, they get "blackjack", which means they win immediately. If the dealer has blackjack, they will reveal their second card, and it will be game over. If both players have blackjack in their first hand, it will be a "two player blackjack", which means the game is tied, "push". If neither player has blackjack in their opening hand, the player will play. If the player busts, the dealer will reveal their second card and win automatically, otherwise the dealer will play. If the dealer busts, the player wins, otherwise they decide the winner by comparing their hand totals; the highest hand total wins.

In this implementation, after the player quits the game, the player's name and average wins will be added to a text file. If their win percentage is higher than all other scores in the file, the game will display, "New High Score!", and display a table containing all players and their recorded scores.

You are required to write a Python program, called ***yourStudentId***`_blackjack.py`, that allows a player to play Blackjack against the computer. You must implement the functions specified in this specification, and then use them to play the game. You will need to import and make use of the provided module, `card_deck.py` and use the `draw_card()` function defined within it. Please **do not** modify or add code to `card_deck.py`.

GAMEPLAY

Our implementation's gameplay is as follows:

- The player and the dealer are dealt an initial hand of two cards each. The player shows both cards. The dealer shows only one card.
- If the dealer has Blackjack (the first two cards dealt total 21 points) and the player does not, the dealer automatically wins.
- If the player has Blackjack (the first two cards dealt total 21 points) and the dealer does not, the player automatically wins.
- If both the player and the dealer have Blackjack (the first two cards dealt total 21 points) then it is a push (draw).
- If neither have Blackjack, the player then plays out their hand.
  Note that the term Blackjack can only be achieved as a result of the first two cards dealt.
- When the player has finished playing out their hand, the dealer (in this case the computer) plays out the dealer's hand.
- During the player's turn, the player is faced with two options either:
  1. Hit (take a card):

     After the initial deal of two cards, a player may choose to receive additional cards as many times as they wish, adding the point value of each card to their hand total.
  2. Stand (end their turn):

     Do not receive any more cards. The player's turn is over.
- The player repeatedly takes a card until, the player chooses to stand, or the player busts i.e., exceeds a total of 21. The player's turn is over after deciding to stand or if they bust.
- Once the player has finished, if the player has not bust, the dealer reveals their hidden second card and plays out their hand. The dealer must hit until they have a point value of 17 or greater.

Rules:

- If the player busts, they lose immediately.
- If the player and the dealer have the same point value, it is called a "push" and neither win the hand (draw).
- An initial two-card deal of 21 (an 11 plus a 10) in the very first hand is called Blackjack and wins the round automatically.
- An Ace counts as eleven (11) points unless this would cause the player/dealer to bust, in which case it is worth one (1) point.
- The dealer **must** hit until they have a point value of 17 or greater.
- The player with the higher hand that does not exceed 21 wins the game!

MODULE CARD_DECK.PY

You **do not** need to write code to create cards or put them into a deck. In fact, you will lose marks if you do!

The `card_deck` module has been provided with this assignment. **Do not** modify or add code to this file. The `card_deck` module contains 1 function that you **must** use: `draw_card()`. The `draw_card()` function returns a card in the form of a list containing 2 strings: the card's value, and the card's suit. Each time you call this function, it will return one of 52 possible cards. The cards will be returned in a random order. For example, the function may return the following:

```
>>> card_deck.draw_card()
['10', 'Hearts']

>>> card_deck.draw_card()
['Jack', 'Spades']

>>> card_deck.draw_card()
['2', 'Clubs']

>>> card_deck.draw_card()
['Ace', 'Diamonds']
```

The deck is created, shuffled, and reset when empty. All these things happen automatically. There are other functions and variables in the `card_deck` module, but you **must only** call the `draw_card()` function.

REQUIREMENTS

Your solution **MUST** use:

- Python 3.8 or later to complete your assignments. Your program MUST run with Python 3.8.
- The supplied `yourStudentId_blackjack.py` file. You must add your solution to this file.
- The supplied `card_deck.py` module and the `draw_card()` function within. You must not modify this file.
- Functions (`display_details, input_name, input_hit_choice, display_hand, get_hand_total, player_play, dealer_play, play_game,` and `add_score`) implemented adhering to the assignment specifications.
- Output that **strictly** adheres to the assignment specifications.
- Good programming practice:
  - You must provide comments to describe your details, program description, academic integrity statement, all functions, and every significant section of code.
  - Meaningful variable names.
  - Consistent indentation.

Your solution **MUST NOT** use:

- `break`, `return`, `quit()`, or `exit()` to stop loops or `continue` statements. *If your loops and if statements are written correctly, you will not need these statements*.
- The `join()` method i.e., `str.join(list)`. *You should use appropriate for loops and if statements instead.*
- Recursion i.e., functions that call themselves. *You should use appropriate while loops instead.*
- List comprehensions or generator expressions. *You should use appropriate for loops and if statements instead.*

STAGES

It is recommended that you develop this part of the assignment in the suggested stages. Many problems in later stages are due to errors in early stages. **Make sure you have finished and thoroughly tested each stage before continuing.**

The following stages of development are recommended:

Stage 1

You will need the `card_deck.py` and the `yourStudentId_blackjack.py` files for this assignment. These have been provided for you. Please download these files from the course website and ensure that they are in the same directory.

Test to ensure that this is working correctly by running your `yourStudentId_blackjack.py` file. You should now see the following output in the Python shell when you run your program:

```
--------- Welcome to Blackjack ---------

---------- See you again soon ----------
```

In your file `yourStudentId_blackjack.py`, you will see the following code:

```
# TODO: Add your academic integrity comment here

import card_deck

# TODO: Define your functions here

def play_game():
    print("--------- Welcome to Blackjack ---------\n")
    # TODO: Write your game code here

if __name__ == '__main__':
    play_game()
    print("---------- See you again soon ----------")
```

Note the first comment, "**`# TODO: Add your academic integrity comment here`**". **<u>Delete</u>** this comment and replace it with your academic integrity statement:

```
#
# File: fileName.py
# Author: your name
# Student ID: your id
# Description: Assignment 2 – place assignment description here...
# This is my own work as defined by Eynesbury
# Academic Misconduct policy.
#
```

Replace the file, author, ID, and description with appropriate information.

Stage 2

Note the second comment, "**`# TODO: Define your functions here`**". **Delete** this comment and replace it with a function definition called `display_details`. This function should take three parameters: `filename`, `author`, and `student_id`. The function should display these details in the format:

```
File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.
```

Replace the highlighted values with your arguments. Your output must match the exact format shown above. The function returns nothing.

Note the third comment, "**`# TODO: Write your game code here`**". **Delete** this comment and replace it with code that calls the `display_details` function with your file name, name, and ID number as arguments.

Note: In this assignment, you will need to define a number of functions (similar to `display_details()`) and then you will need to call these functions from within the `play_game()` function. You should not write any code outside of functions in this assignment. If you do, it will lead to a loss of marks for use of global variables.

Run the code to ensure that it is working. For example, if you called `display_details` with the arguments `("512345_blackjack.py", "Ada Lovelace", "512345")` you should see the following output:

```
--------- Welcome to Blackjack ---------

File   : 512345_blackjack.py
Author : Ada Lovelace
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

---------- See you again soon ----------
```

Call the function with your own details as arguments. Make sure the program runs correctly. Once you have that working, save a new copy of your program as a back-up.

*Note: When developing software, you should always have fixed points in your development where you know your software is bug free and runs correctly. Create a back-up after completing and testing **each stage** of the assignment.*

Stage 4

Write code in the play_game() function to ask the user if they want to play blackjack. Validate this input with a loop that ensures the user may only enter 'y' or 'n'. Display an error message whenever they enter something else. Run the code and ensure your validation loop is working:

```
--------- Welcome to Blackjack ---------

File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

Do you want to play blackjack (y/n): yes
ERROR: Only enter 'y' or 'n'
Would you like to play blackjack (y/n): no
ERROR: Only enter 'y' or 'n'
Would you like to play blackjack (y/n): y

---------- See you again soon ----------
```

Stage 5

Define a function called `input_name()` that has no parameters and returns the user's input name. It should prompt for and take keyboard input for the player's name. It should validate this input in a loop to ensure the name contains no space characters " " and that the length of the name is less than 12. An error message should be displayed whenever the user enters an invalid name.

In the `play_game()` function, if the user wants to play, call the `input_name()`function to take input for their name. Store the returned value in a variable. Run the code to ensure the validation loop is working:

```
--------- Welcome to Blackjack ---------

File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

Do you want to play blackjack (y/n): y
Enter your name: Mahatma Gandhi
ERROR: Must be 1 word and less than 12 characters.
Enter your name: Abdullrahman
ERROR: Must be 1 word and less than 12 characters.
Enter your name: Xu

---------- See you again soon ----------
```

If the user does not want to play, output "Maybe next time...".

```
--------- Welcome to Blackjack ---------

File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

Do you want to play blackjack (y/n): n
Maybe next time...

---------- See you again soon ----------
```

Stage 6

It is time to write code to play the game. Write a loop that runs while the user wants to play. Inside the loop, ask if they want to play again. Every time you take input, follow it with a loop to validate that input. Write code that calculates the number of games played and output this information at the end of the loop. Ensure your output matches the following:

```
...
Do you want to play blackjack (y/n): y
Enter your name: Mike

Do you want to play again (y/n): y

Do you want to play again (y/n): y

Do you want to play again (y/n): no
ERROR: Only enter 'y' or 'n'.
Would you like to play again (y/n): yes
ERROR: Only enter 'y' or 'n'.
Would you like to play again (y/n): n

You played 3 games.

---------- See you again soon ----------
```

Stage 7

In the `play_game()` function, call the `draw_card()` function from the `card_deck` module. Store the result into a list called `dealer_hand`. Dealer_hand should be a list containing another list containing two strings: the value and suit of a random playing card. Call the function 2 more times, storing the results into another list called `player_hand`. Print these lists to ensure the program is working. Your output should look like the following:

```
...
Do you want to play blackjack (y/n): y
Enter your name: Bob

Player hand: [['Jack', 'Hearts'], ['7', 'Spades']]
Dealer hand: [['3', 'Diamonds']]
Do you want to play again (y/n): y

Player hand: [['3', 'Spades'], ['King', 'Spades']]
Dealer hand: [['8', 'Spades']]
Do you want to play again (y/n): y

Player hand: [['Queen', 'Spades'], ['Queen', 'Clubs']]
Dealer hand: [['Jack', 'Diamonds']]
Do you want to play again (y/n): n

You played 3 games.

---------- See you again soon ----------
```

If this does not work, speak to your lecturer.

Stage 8

Implement a function called `display_hand(player_name, hand)`. This function takes a `player_name` (either the user's name or "Dealer") and `hand` (a list of cards) as arguments. It is assumed the argument to the `hand` parameter is either one of the `dealer_hand` or `player_hand` lists created previously. The function does not return anything. It displays the hand to the screen in the following format.

For example, if the arguments to `display_hand` were `'Cathy'` and `[['Queen', 'Spades'], ['9', 'Diamonds']]`:

```
Cathy's hand: Queen of Spades, 9 of Diamonds
```

If the arguments to `display_hand` were `'Dealer'` and `[['Ace', 'Clubs']]`:

```
Dealer's hand: Ace of Clubs
```

Note that the `hand` argument is a ***list*** of ***lists*** of ***strings***. You will need to use a loop to access each list in the list. You may use indexes to access the value and suit.

In the `play_game()` function, instead of printing `dealer_hand` and `player_hand`, call the `display_hand` function with appropriate arguments so that your program produces output in the following format, for example:

```
...
Enter your name: Cathy

Dealer's hand: Jack of Hearts

Cathy's hand: 5 of Spades, Ace of Spades

Do you want to play again (y/n): n

You played 1 game.

---------- See you again soon ----------
```

Stage 9

Define a function called `get_hand_total(hand)` that takes a `hand` as an argument and returns the total point value of all card values in the `hand`.

For example, if the `player_hand` list is...

```
  player_hand = [['10', 'Clubs'], ['7', 'Clubs']]
```

Calling the `get_hand_total(hand)` function like so...

```
  player_total = get_hand_total(player_hand)
```

Will assign the value of `17` to variable `player_total`.

If the `dealer_hand` list is...

```
  dealer_hand = [['Ace', 'Spades']]
```

Calling the `get_hand_total(hand)` function like so...

```
  dealer_total = get_hand_total(dealer_hand)
```

Will assign the value of `11` to variable `dealer_total`.

Some other examples:

- `[['8', 'Spades'], ['7', 'Hearts'], ['10', 'Clubs']]` should return `25`.
- `[['King', 'Spades'], ['Jack', 'Clubs']]` should return `20`.
- `[['Ace', 'Diamonds'], ['8', 'Clubs']]` should return `19`.
- `[['Ace', 'Hearts'], ['Ace', 'Spades'], ['Queen', 'Clubs']]` should return `12`.

Call the `get_hand_total(hand)` function from within the **`display_hand(player_name, hand)`** function to output the hand total in the format:

```
...
Do you want to play blackjack (y/n): y
Enter your name: Jack

Dealer's hand: Jack of Diamonds
Hand Total: (10)

Jack's hand: Jack of Hearts, 7 of Spades
Hand Total: (17)

Do you want to play again (y/n): n

You played 1 game.

---------- See you again soon ----------
```

Stage 10

Include code to check for "Blackjack" on the initial deal of two cards for both the player and dealer. *Note the dealer is dealt another card in secret that is not displayed.*

- If the dealer has Blackjack (the first two cards dealt total 21 points) and the player does not, the dealer automatically wins.
- If the player has Blackjack (the first two cards dealt total 21 points) and the dealer does not, the player automatically wins.
- If both the player and the dealer have Blackjack (the first two cards dealt total 21 points) then it is a push (draw).
- If neither player has Blackjack, the game should continue and the player should play out their hand.

Sample output - Player wins:
```
  Dealer's hand: Queen of Diamonds
  Hand Total: (10)

  Rose's hand: Ace of Spades, 10 of Diamonds
  Hand Total: (21)

  Blackjack! Rose wins!
```

Sample output - Dealer wins:
```
  Dealer's hand: Ace of Spades
  Hand Total: (11)

  Rose's hand: 4 of Diamonds, Queen of Hearts
  Hand Total: (14)

  Dealer's hand: Ace of Spades, King of Spades
  Hand Total: (21)

  Blackjack! Dealer wins!
```

Sample output - Two player blackjack:
```
  Dealer's hand: Ace of Spades
  Hand Total: (11)

  Rose's hand: Ace of Diamonds, Queen of Hearts
  Hand Total: (21)

  Dealer's hand: Ace of Spades, King of Spades
  Hand Total: (21)

  Two player blackjack! -> Push!
```

Sample output - No winner:
```
  Dealer's hand: 9 of Hearts
  Hand Total: (9)

  Rose's hand: Queen of Clubs, Jack of Spades
  Hand Total: (20)
```

Neither player has blackjack. Play should continue and Rose should play out her hand.

Stage 11

Okay, now let's include a few functions to allow the player to play out their hand.

First let's define a function called `input_hit_choice()` that returns the user's choice of 'h' (hit) or 's' (stand). It should prompt for, read, and validate the user's choice to either hit ('h') or stand ('s'). Whenever the user enters an invalid choice, it should display an error message.

Define another function called `player_play(name, hand)` that takes the player's name and list of cards as arguments and returns nothing. In a loop, it should call the input_hit_choice() function to get the user's choice and if the user chooses to hit ('h'), it should deal the player another card. The loop should continue until either the user chooses to stand ('s') or the user's hand total exceeds 21. Display the player’s hand at every step of the loop.

Call the `player_play` function from `play_game()` if neither player has blackjack in their initial hand.

Sample Output - Hit then Stand:
```
  Dealer's hand: Queen of Clubs
  Hand Total: (10)

  Voltron's hand: 2 of Hearts, Ace of Spades
  Hand Total: (13)

  Do you want to hit or stand (h/s): h

  Voltron's hand: 2 of Hearts, Ace of Spades, 10 of Spades
  Hand Total: (13)

  Do you want to hit or stand (h/s): s

  Do you want to play again (y/n): n
```

Sample Output - Invalid input:
```
  Voltron's hand: 3 of Spades, 7 of Diamonds
  Hand Total: (10)

  Do you want to hit or stand (h/s): hit
  ERROR: Must be 'h' or 's'.
  Do you want to hit or stand (h/s): stand
  ERROR: Must be 'h' or 's'.
  Do you want to hit or stand (h/s): s

  Do you want to play again (y/n): n
```

Sample Output - Hit until bust:
```
  Dealer's hand: 8 of Spades
  Hand Total: (8)

  Voltron's hand: 5 of Spades, 6 of Clubs
  Hand Total: (11)

  Do you want to hit or stand (h/s): h

  Voltron's hand: 5 of Spades, 6 of Clubs, 4 of Spades
  Hand Total: (15)

  Do you want to hit or stand (h/s): h

  Voltron's hand: 5 of Spades, 6 of Clubs, 4 of Spades, 2 of Hearts
  Hand Total: (17)

  Do you want to hit or stand (h/s): h

  Voltron's hand: 5 of Spades, 6 of Clubs, 4 of Spades, 2 of Hearts, 9 of Clubs
  Hand Total: (26)

  Do you want to play again (y/n): n
```

Stage 12

Now let's add a function for the dealer's (computer's) turn called `dealer_play(hand)` that takes a hand of cards as an argument and returns nothing. Implement a loop that adds cards to the hand until the hand total is 17 or greater. At each step of the loop, display the dealer's hand and use the input function to ask the user to press "Enter" to continue.

In the `play_game()` function, after the player has played their turn, display the dealer's hand to reveal their face-down card. If the player has not bust, call the `dealer_play` function to allow the dealer (computer) to play out their hand.

Sample output - Dealer plays until bust:

```
  Dealer's hand: Jack of Diamonds
  Hand Total: (10)

  Zelda's hand: 5 of Diamonds, 6 of Hearts
  Hand Total: (11)

  Do you want to hit or stand (h/s): h

  Zelda's hand: 5 of Diamonds, 6 of Hearts, 4 of Clubs
  Hand Total: (15)

  Do you want to hit or stand (h/s): s

  Dealer's hand: Jack of Diamonds, 3 of Hearts
  Hand Total: (13)

  Press "Enter" to continue...

  Dealer's hand: Jack of Diamonds, 3 of Hearts, Queen of Hearts
  Hand Total: (23)

  Press "Enter" to continue...

  Do you want to play again (y/n): n
```

Sample output - Dealer plays until 17 points or more:

```
  Dealer's hand: 4 of Spades
  Hand Total: (4)

  Zelda's hand: 8 of Clubs, King of Spades
  Hand Total: (18)

  Do you want to hit or stand (h/s): s

  Dealer's hand: 4 of Spades, 7 of Clubs
  Hand Total: (11)

  Press "Enter" to continue...

  Dealer's hand: 4 of Spades, 7 of Clubs, 8 of Hearts
  Hand Total: (19)

  Press "Enter" to continue...

  Do you want to play again (y/n): n
```

Stage 13

Add code to determine the winner and display the result to the screen. Between each game print a row of 40 dash characters ('-'). Test your code **thoroughly** to ensure you have the correct result at the end of each game.

Sample output - Player wins:
```
  Dealer's hand: Ace of Spades
  Hand Total: (11)

  Link's hand: 10 of Spades, 10 of Diamonds
  Hand Total: (20)

  Do you want to hit or stand (h/s): s

  Dealer's hand: Ace of Spades, 2 of Spades
  Hand Total: (13)

  Press "Enter" to continue...

  Dealer's hand: Ace of Spades, 2 of Spades, 6 of Diamonds
  Hand Total: (19)

  Press "Enter" to continue...

  Dealer: 19   Link: 20  ->  Link wins!

  ----------------------------------------
```

Sample output - Player busts:
```
  Do you want to play again (y/n): y

  Dealer's hand: 4 of Hearts
  Hand Total: (4)

  Link's hand: Jack of Clubs, Queen of Hearts
  Hand Total: (20)

  Do you want to hit or stand (h/s): h

  Link's hand: Jack of Clubs, Queen of Hearts, 2 of Diamonds
  Hand Total: (22)

  Link bust!
  Dealer: 14   Link: 22  ->  Dealer wins!

  ----------------------------------------
```

Sample output - Players push:
```
  Dealer's hand: 6 of Clubs
  Hand Total: (6)

  Link's hand: 10 of Hearts, 9 of Clubs
  Hand Total: (19)

  Do you want to hit or stand (h/s): s

  Dealer's hand: 6 of Clubs, 3 of Diamonds
  Hand Total: (9)

  Press "Enter" to continue...

  Dealer's hand: 6 of Clubs, 3 of Diamonds, Queen of Hearts
  Hand Total: (19)

  Press "Enter" to continue...

  Dealer: 19   Link: 19  ->  Push!

  ----------------------------------------
```

Stage 14

Add code to calculate how many games were won, lost, and tied. Display this information when the user decides to stop playing. You must format this output as seen below:

```
...
----------------------------------------

Do you want to play again (y/n): n

You played 6 games.
 -> Won:    2
 -> Lost:   3
 -> Tied:   1

Thanks for playing!

---------- See you again soon ----------
```

Stage 15

Time to record the high scores. A file called "highscores.txt" has been provided for you. Each line in the file is a player's name and that player's score separated by a space, for example:

```
Tiffany 37.5
Mike 0.6666666666666666
```

Add a function called `add_score(name, score, filename)` that takes arguments for the player's name, score, and the name of the file to read and write to. The function must read the file to check if the argument score is greater than all other scores in the file. If it is, it should display, "New High Score!", followed by a table containing all names and scores with the player's name and high score at the top. If it is a new high score, the player's name and score should be written into the file on the first line, otherwise they should be written to the file on the last line.

Call this `add_score` function with the player's name, score, and the filename "highscores.txt" as arguments. The player's score can be calculated using the formula:

```
wins/(games-tied)*100
```

Sample Output - New High Score:

```
----------------------------------------

Do you want to play again (y/n): n

You played 4 games.
 -> Won:    3
 -> Lost:   1
 -> Tied:   0

New High Score!

NAME          SCORE
Jack          75.000
Tiffany       37.500
Mike          0.667

Thanks for playing!

---------- See you again soon ----------
```

Sample Output - No High Score:

```
  ----------------------------------------

  Do you want to play again (y/n): n

  You played 2 games.
   -> Won:    0
   -> Lost:   2
   -> Tied:   0

  Thanks for playing!

  ---------- See you again soon ----------
```

Stage 16

Finally, check the sample output and, if necessary, modify your code so that:

- The output produced by your program **EXACTLY** adheres to the sample output provided.
- Your program behaves as described in these specs and as the sample output provided.

SAMPLE OUTPUT

Sample Output - Validate play then quit:

```
  --------- Welcome to Blackjack ---------

  File   : blackjack.py
  Author : Bruce Wayne
  ID     : 512345
  This is my own work as defined by the
  Eynesbury Academic Integrity Policy.

  Do you want to play blackjack (y/n): yes
  ERROR: Only enter 'y' or 'n'
  Would you like to play blackjack (y/n): no
  ERROR: Only enter 'y' or 'n'
  Would you like to play blackjack (y/n): n
  Maybe next time...

  ---------- See you again soon ----------
```

Sample Output - Validate name, hit, and play again:

```
  --------- Welcome to Blackjack ---------

  File   : blackjack.py
  Author : Bruce Wayne
  ID     : 512345
  This is my own work as defined by the
  Eynesbury Academic Integrity Policy.

  Do you want to play blackjack (y/n): y
  Enter your name: Mr Men
  ERROR: Must be 1 word and less than 12 characters.
  Enter your name: Frankenstein
  ERROR: Must be 1 word and less than 12 characters.
  Enter your name: Joker

  Dealer's hand: 4 of Spades
  Hand Total: (4)

  Joker's hand: 3 of Diamonds, 10 of Spades
```

```
Hand Total: (13)

Do you want to hit or stand (h/s): hit
ERROR: Must be 'h' or 's'.
Do you want to hit or stand (h/s): sit
ERROR: Must be 'h' or 's'.
Do you want to hit or stand (h/s): h

Joker's hand: 3 of Diamonds, 10 of Spades, 9 of Spades
Hand Total: (22)

Joker bust!
Dealer: 14   Joker: 22  ->  Dealer wins!

----------------------------------------

Do you want to play again (y/n): yep
ERROR: Only enter 'y' or 'n'.
Would you like to play again (y/n): no
ERROR: Only enter 'y' or 'n'.
Would you like to play again (y/n): n

You played 1 game.
 -> Won:    0
 -> Lost:   1
 -> Tied:   0

Thanks for playing!

---------- See you again soon ----------
```

Sample Output - Win, Lose, Draw, New High Score:

```
--------- Welcome to Blackjack ---------

File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

Do you want to play blackjack (y/n): y
Enter your name: Royal

Dealer's hand: 3 of Diamonds
Hand Total: (3)

Royal's hand: 6 of Clubs, 6 of Spades
Hand Total: (12)

Do you want to hit or stand (h/s): h

Royal's hand: 6 of Clubs, 6 of Spades, 8 of Diamonds
Hand Total: (20)

Do you want to hit or stand (h/s): s

Dealer's hand: 3 of Diamonds, 4 of Clubs
Hand Total: (7)

Press "Enter" to continue...

Dealer's hand: 3 of Diamonds, 4 of Clubs, 10 of Diamonds
Hand Total: (17)

Press "Enter" to continue...

Dealer: 17   Royal: 20  ->  Royal wins!

----------------------------------------

Do you want to play again (y/n): y
```

```
Dealer's hand: 3 of Hearts
Hand Total: (3)

Royal's hand: 2 of Diamonds, Ace of Diamonds
Hand Total: (13)

Do you want to hit or stand (h/s): s

Dealer's hand: 3 of Hearts, 10 of Clubs
Hand Total: (13)

Press "Enter" to continue...

Dealer's hand: 3 of Hearts, 10 of Clubs, 8 of Hearts
Hand Total: (21)

Press "Enter" to continue...

Dealer: 21   Royal: 13  ->  Dealer wins!

----------------------------------------

Would you like to play again (y/n): y

Dealer's hand: 7 of Diamonds
Hand Total: (7)

Royal's hand: King of Diamonds, 9 of Spades
Hand Total: (19)

Do you want to hit or stand (h/s): s

Dealer's hand: 7 of Diamonds, 3 of Diamonds
Hand Total: (10)

Press "Enter" to continue...

Dealer's hand: 7 of Diamonds, 3 of Diamonds, 9 of Diamonds
Hand Total: (19)

Press "Enter" to continue...

Dealer: 19   Royal: 19  ->  Push!

----------------------------------------

Do you want to play again (y/n): n

You played 3 games.
 -> Won:    1
 -> Lost:   1
 -> Tied:   1

New High Score!

NAME         SCORE
Royal        50.000
Tiffany      37.500
Mike         0.667

Thanks for playing!

---------- See you again soon ----------
```

Sample Output - Player bust, Dealer bust:

```
--------- Welcome to Blackjack ---------

File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

Do you want to play blackjack (y/n): y
Enter your name: Rook

Dealer's hand: 8 of Clubs
Hand Total: (8)

Rook's hand: 8 of Spades, 8 of Diamonds
Hand Total: (16)

Do you want to hit or stand (h/s): h

Rook's hand: 8 of Spades, 8 of Diamonds, 10 of Diamonds
Hand Total: (26)

Rook bust!
Dealer: 18   Rook: 26  ->  Dealer wins!

----------------------------------------

Do you want to play again (y/n): y

Dealer's hand: 9 of Hearts
Hand Total: (9)

Rook's hand: King of Diamonds, 4 of Spades
Hand Total: (14)

Do you want to hit or stand (h/s): s

Dealer's hand: 9 of Hearts, 4 of Hearts
Hand Total: (13)

Press "Enter" to continue...

Dealer's hand: 9 of Hearts, 4 of Hearts, 9 of Diamonds
Hand Total: (22)

Press "Enter" to continue...

Dealer bust!
Dealer: 22   Rook: 14  ->  Rook wins!

----------------------------------------

Do you want to play again (y/n): n

You played 2 games.
 -> Won:    1
 -> Lost:   1
 -> Tied:   0

New High Score!

NAME         SCORE
Rook         50.000
Royal        38.889
Tiffany      37.500
Mike         0.667

Thanks for playing!

---------- See you again soon ----------
```

Sample Output - Player blackjack, Dealer blackjack, Double blackjack:

```
--------- Welcome to Blackjack ---------

File   : blackjack.py
Author : Bruce Wayne
ID     : 512345
This is my own work as defined by the
Eynesbury Academic Integrity Policy.

Do you want to play blackjack (y/n): y
Enter your name: Jack

Dealer's hand: 3 of Clubs
Hand Total: (3)

Jack's hand: Ace of Spades, Queen of Hearts
Hand Total: (21)

Blackjack! Jack wins!

----------------------------------------

Do you want to play again (y/n): y

Dealer's hand: King of Hearts
Hand Total: (10)

Jack's hand: 10 of Clubs, 8 of Spades
Hand Total: (18)

Dealer's hand: King of Hearts, Ace of Clubs
Hand Total: (21)

Blackjack! Dealer wins!

----------------------------------------

Do you want to play again (y/n): y

Dealer's hand: King of Spades
Hand Total: (10)

Jack's hand: 10 of Hearts, Ace of Diamonds
Hand Total: (21)

Dealer's hand: King of Spades, Ace of Clubs
Hand Total: (21)

Two player blackjack! -> Push!

----------------------------------------

Do you want to play again (y/n): n

You played 53 games.
 -> Won:    21
 -> Lost:   31
 -> Tied:   1

Thanks for playing!

---------- See you again soon ----------
```

SUBMISSION DETAILS

You are required to submit the assignment via Moodle. Make sure your files are included in a zip file. The zip file should be called `yourIdAss2.zip`. For example: `503123Ass2.zip`

Ensure that the following files are included in your submission:

- `yourStudentId_blackjack.py` *(renamed with your ID)*
- `card_deck.py`
- `highscores.txt`

`yourStudentId_blackjack.py` must include the following comments, modified to include your details:

```
#
# File: fileName.py
# Author: your name
# Student ID: your id
# Description: Assignment 2 – place assignment description here...
# This is my own work as defined by Eynesbury
# Academic Misconduct policy.
#
```

Assignments that do not contain these details may not be marked. It is expected that students will make back-up copies of all assignments and be able to provide these if required. Students may also be expected to explain parts of their assignment to the marking lecturer or rewrite parts of it to show their full understanding of the work.

EXTENSIONS AND LATE SUBMISSIONS

There will be **no** extensions/late submissions for this course without one of the following exceptions:

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. **Please note** if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Coordinator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.

2. An Eynesbury counsellor contacts the Course Coordinator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.

3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.

4. Military obligations with proof.

Applications for extensions must be lodged with the Course Coordinator before the due date of the assignment

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension.

Academic Misconduct

Deliberate academic misconduct, such as plagiarism, is subject to penalties such as receiving 0 for the assignment or being expelled from the college.

- **DO NOT** share your code or images of your code with others.
- **DO NOT** copy code found on the internet or from any other source.
- **DO NOT** allow others to write your code for you.

Suspicious submissions will be investigated. Information about Academic integrity can be found in the "Policies and Procedures" section of the Eynesbury website.

Marking Criteria

NAME:	**Problem Solving and Programming (COMP 1039) Assignment 2 - Weighting: 15%**		
OUTPUT	**SPECIFICATION**	**MARK**	**MAX MARK**
`File   : blackjack.py` `Author : Bruce Wayne` `ID     : 512345` `This is my own work as defined by the Eynesbury Academic Integrity Policy.`	`display_details()` defined correctly		**4**
	Function called correctly		**4**
	Output formatted correctly		**4**
`Do you want to play blackjack (y/n): INVALID` `ERROR: Only enter 'y' or 'n'`	Play validation loop		**1**
`Would you like to play blackjack (y/n): y` `Enter your name:`	Play 'y'		**1**
`Do you want to play blackjack (y/n): n` `Maybe next time...`	Play 'n'		**1**
`Enter your name: John Wayne`	`input_name()` defined correctly and returns		**2**
`ERROR: Must be 1 word and less than 12 characters.`	Validates space		**1**
`Enter your name: Abdullrahman` `ERROR: Must be 1 word and less than 12 characters.`	Validates 12 characters		**1**
`Enter your name: Joker`	inputs correctly		**1**
	`display_hand()` defined correctly		**3**
`Dealer's hand: Ace of Clubs`	`display_hand()` called from `play_game()` correctly		**4**
`Hand Total: (11)`	Dealer shows 1 card		**1**
`Joker's hand: 4 of Diamonds, King of Spades`	Player shows 2 cards		**2**
`Hand Total: (14)`	Calls `get_hand_total()` correctly and formats output		**3**
`Joker's hand: 4 of Diamonds, King of Spades, Ace of Spades`	`get_hand_total()` defined correctly		**2**
`Hand Total: (15)`	Calculates total correctly		**4**
	`player_play()` defined correctly		**3**
	`player_play()` calls other functions		**4**
`Do you want to hit or stand (h/s): sit`	`input_hit_choice()` defined correctly		**2**
`ERROR: Must be 'h' or 's'.`	Validates 'h' or 's'		**2**
`Dealer's hand: 5 of Clubs, Ace of Clubs` `Hand Total: (16)`	Dealer reveals 2nd card		**1**
`Press "Enter" to continue...`	Press enter between each dealer turn		**2**
`Dealer's hand: 5 of Clubs, Ace of Clubs, 10 of Spades`	dealer_play() defined correctly		**2**
`Hand Total: (17)`	stops after 17		**1**
`Dealer: 17   Joker: 15  ->  Dealer wins!`	Formats results correctly		**3**
`Dealer wins!`	Dealer wins!		**1**
`Joker wins!`	Player wins!		**1**
`Push!`	Push!		**1**

`Dealer bust!`	Dealer bust!		**1**
`Joker bust!`	Player bust!		**1**
`Blackjack! Dealer wins!`	Blackjack! Dealer wins!		**1**
`Blackjack! Joker wins!`	Blackjack! Player wins!		**1**
`Two player blackjack! -> Push!`	Two player blackjack! -> Push!		**1**
`----------------------------------------`	games separated by 40 -'s		**1**

`Do you want to play again (y/n): INVALID`
`ERROR: Only enter 'y' or 'n'.`
	Play again validation loop		**2**

`Would you like to play again (y/n): y`	Play again		**1**
`Do you want to play again (y/n): n`	Quit game		**1**
`You played 6 games.`	Formatted and calculated correctly		**2**
`-> Won:    3`	Formatted and calculated correctly		**2**
`-> Lost:   2`	Formatted and calculated correctly		**2**
`-> Tied:   1`	Formatted and calculated correctly		**2**
	add_score() defined correctly		**4**
	add_score() called correctly		**4**
`New High Score!`	Correctly identifies high score		**1**

`NAME         SCORE`
`Joker        75.000`
`Jack         50.000`
	Correctly formats table		**3**

	Correctly writes file top score		**1**
	Correctly writes file bottom score		**1**
`Thanks for playing!`	Thanks for playing!		**1**
	Consistent comments		**2**
	Meaningful variable names		**2**
	Consistent indentation		**1**
	Inconsistent blank lines in output		**-2**
	`Break`, `return`, `quit()`, `exit()`, `continue`, `join()`, recursion, list comprehensions, or generator expressions...		**-2** **PER USE**
	Submitted incorrectly		**-10**
COMMENTS:	**TOTAL**		**100** **MARKS**