Objective

You will write a command-line version of a single game of Blackjack as described below. The only interaction point is whether the player should hit or stand; all else is handled by the program. Hand scores should be present when all card values are known (that is, everywhere except for the dealer when one of their cards is hidden).

In addition to your code file(s), also send a README that contains:

- Instructions on how to run your program, including any dependencies we would need to install
- What assumptions/choices you made to overcome any lack of clarity you found in the instructions
- What you did well on this project
- Rationale on design choices, algorithmic decisions made
- Tradeoffs you encountered while programming and how you resolved them
- What you would improve on this project given more time
- What manual tests you ran on the code
- How to run any automated tests you created (if they exist)

Limit yourself to two hours to write the code for this exercise. Take all the time you need to write the README as we want to understand your thought processes as well as your code. We **strongly** recommend using a scripting language to increase your odds of completing this in the time allotted - Python or Javascript are our favorites.

We expect runnable, testable code. You'll be graded on the viability, readability, and testability of this code along with the details provided in the README. Take some time to write tests for your code. If you do not have enough time to write tests, put into your README how you would test your code.

When you are done, reply to this email with a zipped folder containing your code, README, and any test files you created. We'll do our best to get you a response back ASAP! If you've not heard back from us within one week of sending your response, check in so we're all on the same page.

Thanks, and good luck!

Blackjack Problem Setup

You'll spend the next two hours on a command-line Blackjack program. It's OK if you've not played before; all the information you need to know about the game is on this page. If at any point you are unclear or unsure about part of the instructions, add a comment in your code documenting what point you were confused about and what choice you made to resolve it.

Note that this game is a simplified version of the well-known game. Please carefully read the rules below before starting to code.

Blackjack is a card game. For our purposes, we'll use a single standard 52-card deck. This deck has:

- "number cards" with a face value of 2-10, worth the number on the card
- "face cards" (King, Queen, Jack), each worth 10 points
- "aces" (Ace), worth 1 or 11 points. We'll get to that later.

The above cards constitute a "Suit" in a deck. A deck consists of four suits. The suit names are unimportant, and can be ignored for the purposes of this game. The important thing is that each deck has four twos, four kings, four aces, and so on.

```
Deck: 4 x [2 - 10, J, Q, K, A]
           |___|   |_____|  |
             #       10    1,11
```

All 52 cards should be present in the deck at the start of the game. The cards dealt should be randomized during play - the cards dealt should not be predictable.

For our Blackjack game, there are two players: the human (you) and the dealer (controlled by the computer).

Definitions

Hand: a collection of cards that a player owns for the duration of the game.

Hit: add another card to a player's hand

Stand: add no more cards to a player's hand; stop playing a turn; proceed to the next phase of the game.

Bust: more than 21 points in a player's hand; the player's turn is instantly over; the player has lost.

Winning

Each player's objective is to get as close to 21 points as possible without going over.

The human wants to outscore the dealer, despite not knowing the dealer's score.

The dealer plays by a simple rule: hit until the hand's score is greater than or equal to 17 ("stand on 17").

For our game there is no betting, no splitting, no doubling down. This is a simple game.

Scoring

Scoring is just adding the points of a hand together. For all cards but the ace, this is trivial. Say the player was dealt:

```
 6 Q
---
6 + 10 (remember, all face cards are 10 points)
16
```

At this point, Let's say the player hits and gets a seven.

```
 6 Q 7
-----
6 + 10 + 7
23
```

23 is over 21 so they would have busted, and the game would be over.

Let's instead say that they got a four.

```
 6 Q 4
-----
6 + 10 + 4
20
```

20 is a fantastic place to hold, so the player would likely stand (though it's still their choice).

The intricacies of Ace scoring

An ace can be worth one or eleven points. At any point there is a single "optimal" value for that card. Your program will determine and report that value. We'll explore this with the demonstration hand of 8 A.

```
 8 A
-------------
8 1      8 11
8 + 1    8 + 11
9        19
```

The optimal value for the ace is always the one that creates the greatest non-busted hand value. In this case, that is an 11. If a player decides to hit at this point, they will likely bust with the ace valued at 11. Fortunately, the value is re-assessed with every card dealt. Therefore, if a 7 is dealt:

```
 8 A 7
-------------------
8 1 7         8 11 7
8 + 1 + 7     8 + 11 + 7
16            26
```

For this hand the 16 is obviously preferable to losing the game. Your algorithm should present only that value.

Let's look at multiple aces in the hand: with a starting hand of A A there are three possible hand values: `2 (1 + 1), 12 (11 + 1), or 22 (11 + 11)`, which is a bust. What happens when another ace is added?

`[A, A, A] = 3, 13, 23, or 33 => pick 13.` Note that any hand with more than one ace as an 11 will be a bust.

A sample hand

The game starts with dealing cards. The dealer is dealt two cards: one face up, one face down. The human is then dealt two cards, both face up. Your code should present this in a manner similar to the following:

```
 Dealer has: # ? = ?
Player has: # # = #
```

Where # are known values and ? are unknown values.

Example program output:

```
 Dealer has: J ? = ?
Player has: 7 A = 18
```

The number after the equals sign is the hand's "best" value, as described above.

At this point, the human player plays their hand. To do so, they are asked by the program whether to Hit or Stand. The player can continue to hit as many times as they wish until they Stand or Bust. When the player chooses to stand their turn is over; they have no further opportunity to get more cards. If at any point the player's score goes over 21, they have busted and lost the game.

Example program output:

```
 Dealer has: 3 ? = ?
Player has: 6 9 = 15
Would you like to (H)it or (S)tand? H

Player has: 6 9 10 = 25
Player busts with 25
Dealer wins
```

If at any point the player gets exactly 21, they instantly win and the game is over. Be sure to point out that the player won with a blackjack!

Example program output:

```
 Dealer has: 3 ? = ?
Player has: 10 A = 21
Player wins!
Blackjack!
```

Once the player has stood, the dealer's turn starts. The dealer's hidden card (denoted by ?) is revealed. The dealer then plays by a simple set of rules: Hit until the hand's score is greater than or equal to 17. If at any time the dealer busts the game is over; the player has won. Otherwise, at some point the dealer will stand. At that point whoever has the highest score wins the game. If both hands have the same score the result is a tie. At this point the game is done, the program can exit.

Example program output:

```
 Dealer has: K ? = ?
Player has: A 5 = 16
Would you like to (H)it or (S)tand? H

Dealer has: K ? = ?
Player has: A 5 4 = 20
Would you like to (H)it or (S)tand? S

Player stands with: A 5 4 = 20

Dealer has: K 4 = 14
Dealer hits
Dealer has: K 4 A = 15
Dealer hits
Dealer has: K 4 A 2 = 17
Dealer stands

Player Wins!
A 5 4 = 20 to Dealer's K 6 A = 17
```

Gameplay Overview

1. Deal initial cards (two cards to each player)
2. Display initial hands (hiding dealer's second card and score)
3. Prompt user (Hit or Stand?)
   - Hit: add card to hand (check if busted)
   - Stand: end turn
   - show updated hand and value
   - repeat until player has stood, won (score == 21), or busted (score > 21)
4. Dealer plays (if player has neither busted nor won)
   - print dealer's full hand, score
   - dealer keeps hitting until score >= 17
5. Decide and report the winner, including hands and scores where relevant