# Project 1

MPCS 51042 - Python Programming - Fall 2021

Deadline: November 2 at 5:00 pm CDT

In this assignment, you will write a simple text-based blackjack game. Part of the basic class structure is specified, but you have a great deal of freedom for the rest of the implementation. Proper documentation will be very important for your evaluation.

## 1 How to Play Blackjack

### 1.1 How to Play Blackjack In Real Life

This video ($< 3$ minutes) demonstrates simple rules for humans to play blackjack. Your program will use additionally simplified rules (see next section).

https://www.youtube.com/watch?v=eyoh-Ku9TCI

### 1.2 How to Play Blackjack In Your Program

Your program will follow these simplified rules. Note that there is no betting. Instead, at the end of the round, the player or dealer with the highest score is declared the winner.

1. For each round:
   (a) The dealer shuffles the deck.
   (b) The dealer gives 1 card face-up to all players; and 1 card face-up to themself.
   (c) The dealer gives 1 card face-up to all players; and 1 card face-down to themself.
   (d) For each player's turn:
      i. The player decides to "hit" (receive one more card from dealer) or "stand" (receive no more cards and immediately end their turn). If the player hits and their new score is $> 21$, then they "bust", meaning that their turn immediately ends and they immediately lose.
      ii. Repeat until the player's turn ends, either because they stand or bust.
   (e) For the dealer's turn:
      i. The dealer reveals their face-down card.
      ii. If the dealer's score is $\geq 17$, they must stand.
      iii. If the dealer's score is $< 17$, they must continue to deal cards to themself until their score is $\geq 17$. If their score ever becomes $> 21$, they bust and immediately lose.
   (f) Declare the winner of this round. The winner is the player or dealer with the highest score.
   (g) Prompt the human players if they want another round.

# 2 Program Requirements

## 2.1 User Interface

This will be an entirely text-based game. The on-screen prompts should provide enough information for a user to play the game without referring to any outside sources (like a README). The user will respond with simple text prompts such as entering "hit" or "stand".

## 2.2 Class Structure

Your program must have at least these classes and methods. You can add any additional methods, attributes, and/or classes. You can break this up into multiple modules, if you like.

- An `Game` class with the following methods:
  - `__init__()` takes two parameters: the number of human players (default 1) and the number of computer players (default 1).
  - `play()` begins the game.
- A `Player` abstract base class. It must inherit from `abc.ABC` and use `@abstractmethod` where appropriate. As far as possible, specify the common behavior of both human and computer players.
- A `HumanPlayer` class, which is an implementation of `Player`. It should interactively prompt the user when appropriate.
- A `ComputerPlayer` class, which is another implementation of `Player`. It plays the game automatically and displays its actions to the human users. You will not be evaluated by how well it plays, but feel free to get creative!

## 2.3 How your Program is Run

Your executable program should be in a source file named `blackjack.py`. It should be run on the command line without any command line arguments. It should run a game with 1 human player and 1 computer player (in addition to the dealer). Including this code in your source file will work:

```
if __name__ == "__main__":
    g = Game()
    g.play()
```

## 2.4 Documentation

You **must** fully document your code according to one of these styles.

- NumPy style: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_numpy.html
- Google style: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html
- Sphinx style: https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html

### 2.5 Optional Extra Credit

For 10 points extra credit (out of the 200 possible points), you may briefly present your object model in class during Week 5. Your implementation does not have to be complete; you will get full credit for presenting whatever you have finished. About 5 minutes is sufficient. Some possible ways to present are:

- Some simple, itemized lists of your classes and methods, with descriptions.
- A UML diagram.
- Some Sphinx docs, generated from your in-code docstrings.

## 3 Some Design Choices

Here are a few questions you might consider when you're designing your program. There are no right or wrong answers to these.

- Are the dealer's actions encapsulated into the `Game` class? Or is there a separate `Dealer` class?
- Your `Game` will probably contain one or more collections of `HumanPlayer`s and `ComputerPlayer`s. Can all `HumanPlayer`s and `ComputerPlayer`s be stored the same collection? Does the `Game` have to treat `HumanPlayer`s and `ComputerPlayerss` differently? Or can `Game` treat them all as instances of the base class, `Player`?
- The human user can see all the face-up cards, including the dealer's and the other players'. Is it also necessary for the `ComputerPlayer` to "see" the dealer's and other players' face-up cards?