## Stage 1: Group Scoring

Write a function **`comp10001go_score_group()`** which takes a single argument:

- **`cards`**, a list of cards (each a 2-element string, where the first letter is the card value and the second letter is the card suit, e.g. **`'3H'`** for the 3 of Hearts)

Your function should return an integer score for the group, based on the following:

- Each card is scored based on its value, whereby each number is scored as that number, Jacks, Queens, and Kings are scored 11, 12, and 13, resp., and Aces are scored as 20
- If the group is a valid $N$**-of-a-kind** (i.e. there are 2 or more cards of the same non-Ace value), the score is that value multiplied by $N!$ (i.e. $N$ factorial, which is the product $N \times (N-1) \times \ldots \times 1$)
- If the group is a valid **run** (i.e. a set of 3 or more cards, starting from the lowest-valued non-Ace card, and ending with the highest-valued non-Ace card, forming a continuous sequence in terms of value, and alternating in colour; note that Aces can act as wilds in terms of value — but not colour — as a mid-element of a run), the score is the sum of the values of the component cards, with Aces scored as the value of the card they stand in for. Note that the specific ordering of **`cards`** is not significant in runs, i.e. **`['2C', '3D', '4S']`** and **`['4S', '2C', '3D']`** both make up the same run.
- If the group is a singleton card or doesn't form a valid $N$-of-a-kind or run, it should be scored as the negative sum of the scores of the individual cards (scoring Aces as 20)

Example function calls are as follows:

```
>>> comp10001go_score_group(['2C'])
-2
>>> comp10001go_score_group(['2C', '2S'])
4
>>> comp10001go_score_group(['4C', '4H', '4S'])
24
>>> comp10001go_score_group(['4C', '4H', '3S'])
-11
>>> comp10001go_score_group(['4C', '4H', 'AS'])
-28
>>> comp10001go_score_group(['KC', 'KH', 'KS', 'KD'])
312
>>> comp10001go_score_group(['2C', '3D', '4S'])
9
>>> comp10001go_score_group(['4S', '2C', '3D'])
9
>>> comp10001go_score_group(['2C', '3D', '4H'])
```

## Question 2: Group Validation

Write a function **`comp10001go_valid_groups()`** which takes a single argument:

- **`groups`**, a list of groups, each of which is a list of cards (following the same definition as Q1)

Your function should return a Boolean indicating whether all groups are valid or not (i.e. a singleton card, a valid N-of-a-kind or a valid run). Note that the function may be used to validate a grouping of partial discards or the full set of discards, i.e. the total number of cards in **`groups`** will be between 0 and 10.

Example function calls are as follows:

```
>>> comp10001go_valid_groups([['KC', 'KH', 'KS', 'KD'], ['2C']])
True
>>> comp10001go_valid_groups([['KC', 'KH', 'KS', 'AD'], ['2C']])
False
>>> comp10001go_valid_groups([['KC', 'KH', 'KS', 'KD'], ['2C', '3H']])
False
>>> comp10001go_valid_groups([])
True
```

## Question 3: Play and Group

The third question requires that you implement the two functions that are called in the tournament: (1) **`comp10001_play`**, which is used to select a discard over the 10 turns of a game; and (2) **`comp10001_group`**, which is used to group the discards into groups for scoring. We combine these together into a single question in Grok as a means of validating that you have a complete player that is qualified to enter the tournament. Note that in each case, we provide only a single test case (and no hidden test cases) for two reasons: (1) there are very few game states where there is a single possible option to either play a discard or group the discards, and testing relies on there only being one possible output; and (2) the real testing occurs in *simulation* mode in the actual tournament, in the form of random games against other players. On validation of implementations of each of the two functions, you will be given the option to submit your player to the tournament.

First, write a function **`comp10001go_play()`** which takes three arguments:

- **`discard_history`**, a list of lists of four cards, each representing the discards from each of the four players in preceding turns (up to 9 turns) in sequence of player number (i.e. the first element in each list of four cards is for Player 0, the second is for Player 1, etc.). Note that the list is sequenced based on the turns, i.e. the first list of four cards corresponds to the first turn, and the last list of four cards corresponds to the last turn.
- **`player_no`**, an integer between 0 and 3 inclusive, indicating which player is being asked to play. **`player_no`** can also be used to determine the discards for that player from **`discard_history`** by indexing within each list of four cards.
- **`hand`**, a list of cards held by the player.

Your function should return a single card to discard from **`hand`**.

An example function call is as follows:

```
>>> comp10001go_play([['0S', 'KH', 'AC', '3C'], ['JH', 'AD', 'QS', '5H'],
['9C', '8S', 'QH', '9S'], ['8C', '9D', '0D', 'JS'], ['5C', 'AH', '5S',
'4C'], ['8H', '2D', '6C', '2C'], ['8D', '4D', 'JD', 'AS'], ['0H', '6S',
'2H', 'KC'], ['KS', 'KD', '7S', '6H']], 3, ['QC'])
'QC'
```

Second, write a function **`comp10001go_group()`** which takes two arguments:

- **`discard_history`**, a list of lists of four cards, each representing the discards from each of the four players in preceding turns in sequence of player number (i.e. the first element in each list of four cards is for Player 0, the second is for Player 1, etc.). Note that the list is sequenced based on the turns, i.e. the first list of four cards corresponds to the first turn, and the last list of four cards corresponds to the last turn. Additionally note that the number of turns contained in **`discard_history`** will always be 10.
- **`player_no`**, an integer between 0 and 3 inclusive, indicating which player is being asked to play. **`player_no`** can also be used to determine the discards for that player from **`discard_history`** by indexing within each list of four cards.

Your function should return a list of lists of cards based on the discard history of **`player_no`**, to use in scoring the player. Note that the grouping of cards represented by the output must be valid (i.e. each list of cards must be a singleton card, or a valid N-of-a-kind or run), but that the ordering of cards within groups, and the ordering of groups is not significant.

An example function call is as follows:

```
>>> comp10001go_group([['0S', 'KH', 'AC', '3C'], ['JH', 'AD', 'QS', '5H'],
['9C', '8S', 'QH', '9S'], ['8C', '9D', '0D', 'JS'], ['5C', 'AH', '5S',
'4C'], ['8H', '2D', '6C', '2C'], ['8D', '4D', 'JD', 'AS'], ['0H', '6S',
'2H', 'KC'], ['KS', 'KD', '7S', '6H'], ['JC', 'QD', '4H', 'QC']], 3)
[['3C'], ['5H'], ['9S'], ['JS'], ['4C'], ['2C'], ['AS'],
```

## Question 4: Optimisation

The final question is for bonus marks, and is deliberately quite a bit harder than the four basic questions (and the number of marks on offer is, as always, deliberately not commensurate with the amount of effort required). Only attempt this is you have completed the earlier questions, and are up for a challenge!

Write a function **`comp10001go_best_partitions()`** which takes a single argument:

- **`cards`**, a list of up to 10 cards

Your function should return a list of list of lists of cards, representing the groupings of cards that score the most points from **`cards`**. Note that the ordering of the groupings is not significant, and neither is the ordering of the groups within a grouping, nor the order of cards within a group.
One area of particular focus with this question is efficiency: there are strict time limits associated with running your code over each example, that you must work within, or you will fail the test. Good luck!

Example function calls are as follows:

```
>>> comp10001go_best_partitions(['0H', '8S', '6H', 'AC', '0S', 'JS', '8C',
'7C', '6D', 'QS'])
[[['AC'], ['0H', '0S'], ['JS'], ['8S', '8C'], ['7C'], ['6H', '6D'],
['QS']]]
>>> comp10001go_best_partitions(['9D', '2S', '4D', '4H', '6D', 'AH', '2C',
'JH', '3C', '9H'])
[[['4D', '4H'], ['6D'], ['AH'], ['2S', '2C'], ['JH'], ['3C'], ['9D',
'9H']]]
>>> comp10001go_best_partitions(['3C', '5H', '9S', 'JS', '4C', '2C', 'AS',
'KC', '6H', 'QC'])
[[['3C'], ['5H'], ['9S'], ['JS'], ['4C'], ['2C'], ['AS'], ['KC'], ['6H'],
['QC']]]
>>> comp10001go_best_partitions(['0D', 'AS', '5C', '8H', 'KS', 'AH', 'QH',
'AC'])
[[['AS', '5C', '8H', 'AH'], ['0D', 'KS', 'QH', 'AC']], [['0D', 'AS', 'KS',
'QH'], ['5C', '8H', 'AH', 'AC']]]
```