# M6: Bin Packing Algorithm Assignment

## Instructions

Design and write an algorithm in Python for solving the Bin Packing Problem by doing the following:

- ***Begin with the `binpack-template.py` file and rename it `BinpackingDev.py`.***
- Implement your algorithm in the Python program named `BinpackingDev.py` in the `binpack()` function.
- Use `BinpackingDev.py` to test your algorithm for the problems in the accompanying JSON data file.

### *Guidance*

Details on the `binpack()` function:

- Input arguments:
  - `items`: a dictionary of the items to be place in bins, with:
    - keys representing the item ID values (integers)
    - values representing the item volumes
      - e.g., for example: {0: 2, 1: 5…}
    - The code provided in `BinpackingDev.py` already extracts this information from the JSON data file and inputs it into `binpack()`.
  - `bin_cap`: This value represents the volume capacity of each (identical) bin. Its value is already set in the `binpacking.py` program. You need not make any changes.
- Output parameters:
  - `myUsername`: your W&M username in order to identify the output as your work)
  - `nickname`: a string variable indicating how you would like to be identified on a leaderboard showing the scores of everybody in class, which will be visible to your classmates. If you choose not to be identified on the leaderboard, you may return an empty string, that is, `''`.
  - `items_to_pack` : a list of lists, where each sublist indicates the dictionary keys for the items assigned to one bin
    - Each sublist contains the item ID numbers for the items contained in a bin. For example, this list of lists—[[2,6,8,15],[1,11,7,9],…]—indicates that the 0th bin contains items with these IDs, [2,6,8,15], and the 1st bin contains items with these IDs: [1,11,7,9], etc.
    - Take care to (1) not overload the bins, (2) place all items in one bin, and (3) not assign items to more than one bin.

### *Submission*

- Once you have completed developing your algorithm, submit `BinpackingDev.py` via Canvas.