COMP226

Assignment 1

Vladimir Gusev

vladimir.gusev@liverpool.ac.uk

These slides **a1_slides.pdf** are an **abridged** (shortened) version of the main handout **a1_wsheet.pdf**.

Limit Order Books

Continuous Assessment Number	1 (of 2)
Weighting	15%
Assignment Circulated	Sunday 26 February 2023
Deadline	21:00 Tuesday 14 March 2023
Submission Mode	CodeGrade assignment on Canvas
Learning Outcomes Assessed	Have an understanding of market microstructure and its impact on trading.

Goal of Assignment	Reconstruct a limit order book from order messages; compute quantities based on the limit order book
Marking Criteria	**Pre-deadline** visible CodeGrade tests of correctness of 6 functions (**70%**); **Post-deadline** CodeGrade tests of correctness for 4 "extra" functions (**30%**)
Submission necessary in order to satisfy module requirements	No
Expected time taken	Roughly 8-12 hours

Pre vs. post deadline tests

- 70% of marks are available via **visible pre-deadline tests** on CodeGrade for **6 functions**. **If your code passes all the tests you get the full 70%** for this part of the assignmnent
- You can **submit as many times as you want**, and use the CodeGrade feedback to improve your mark for this part
- 30% of marks are for passing **post-deadline tests** for **4 extra functions** **but** **these marks are only available if you got all 70% for the pre-deadline tests**

Code/data zip handout

- **Download** `comp226_a1.zip`
- **Unzip** `comp226_a1.zip`
- This will yield a directory called `comp226_a1`

```
comp226_a1
■■■ common.R
■■■ input
■   ■■■ book_1.csv
■   ■■■ book_2.csv
■   ■■■ book_3.csv
■   ■■■ empty.txt
■   ■■■ message_a.txt
■   ■■■ message_ar.txt
■   ■■■ message_arc.txt
■   ■■■ message_ex_add.txt
■   ■■■ message_ex_cross.txt
■   ■■■ message_ex_reduce.txt
■   ■■■ message_ex_same_price.txt
■■■ main.R
■■■ output
■   ■■■ book_1-message_a.out
■   ■■■ book_1-message_ar.out
■   ■■■ book_1-message_arc.out
■   ■■■ book_2-message_a.out
■   ■■■ book_2-message_ar.out
■   ■■■ book_2-message_arc.out
■   ■■■ book_3-message_a.out
■   ■■■ book_3-message_ar.out
■   ■■■ book_3-message_arc.out
■■■ template.R

2 directories, 23 files
```

The contents are explained on the next slide..

Code/data zip handout

- **main.R**: **the script that you should call**, examples below (do not edit it)
- **common.R**: **provides a range of fully implemented functions** (do not edit it)
- **template.R**: **code template** that contains 10 empty functions that you need to complete
- **input**: subdirectory that contains two types of **input files**, initial book files and message files
- **output**: subdirectory that contains **sample output** that allows you to check your code implementations

Ex: using main.R with template.R

```
$ Rscript main.R template.R input/book_1.csv input/empty.txt
$ask
  oid price size
1   a   105  100

$bid
  oid price size
1   b    95  100

Total volume:
Best prices:
Mid-price:
Spread:
```

`input/book_1.csv` is the initial book, `input/empty.txt` is the message file (empty in this case)

main.R

```r
options(warn=-1)
args <- commandArgs(trailingOnly = TRUE); nargs = length(args)
log <- (nargs == 4) # TRUE is there are exactly 4 arguments

arg_format <- "<--log> <solution_path> <book_path> <messages_path>"

if (nargs < 3 || nargs > 4)  # check that there are 3 or 4 arguments
    stop(paste("main.R has 3 required arguments and 1 optional flag:", arg_format))

if (nargs == 4 && args[1] != "--log") # if 4 check that --log is the first
    stop(paste("Bad arguments format, expected:", arg_format))

solution_path <- args[nargs-2]
book_path     <- args[nargs-1]
messages_path <- args[nargs]

if (!all(file.exists(c(solution_path, book_path, messages_path))))
    stop("File does not exist at path provided.")

source(solution_path); source("common.R") # source common.R from pwd

book <- book.load(book_path)
book <- book.reconstruct(data.load(messages_path), init=book, log=log)
book.summarise(book)
```

main.R

- checks the command line arguments are ok
- assigns them to variables
- sources `common.R` and the file at `solution_path`
- loads an initial book
- reconstructs the book according to the messages
- prints out the book
- prints out the book stats

Rscript from Rstudio

```
Rscript main.R template.R input/book_1.csv input/empty.txt
```

- In R studio, you can call Rscript from the "terminal" tab (as opposed to the "console")

- On Windows, use Rscript.exe not Rscript:

```
Rscript.exe main.R template.R input/book_1.csv input/empty.txt
```

70%: 6 functions to implement

Order book stats:

1. `book.total_volume <- function(book)` **[5%]**
2. `book.best_prices <- function(book)` **[5%]**
3. `book.midprice <- function(book)` **[5%]**
4. `book.spread <- function(book)` **[5%]**

Reconstructing the limit order book:

5. `book.reduce <- function(book, message)` **[15%]**
6. `book.add <- function(book, message)` **[35%]**

input/book_1.csv

```
oid,side,price,size
a,S,105,100
b,B,95,100
```

oid	side	price	size
a	S	105	100
b	B	95	100

- `oid` (order id) is used to process (partial) cancellations of orders that arise in "reduce" messages
- `side`: 'B' for a buy/bid; 'S' for a sell/ask order
- `price` and `size` are self-explanatory

Order book stats

- `book.total_volumes` should return **a list with named elements**, `bid` and `ask` where `bid` (`ask`) should be the total volume in the bid (ask) book
- `book.best_prices` should return **a list with two named elements**, `bid` and `ask` where `bid` (`ask`) should be the best bid (ask) price
- `book.midprice` should return the midprice
- `book.spread` should return the spread

The functions can be tested with an empty message file (you do not need to have implemented `book.add` or `book.reduce`)

Expected output

```
$ Rscript main.R solution.R input/book_1.csv input/empty.txt
$ask
  oid price size
1   a   105  100

$bid
  oid price size
1   b    95  100

Total volume: 100 100
Best prices: 95 105
Mid-price: 100
Spread: 10
```

book.add and book.reduce

For the 5th and 6th functions, `book.add` and `book.reduce`, you need to understand the format of messages...

Message format

- message files contain one message per line (terminated by a single linefeed character, '`\n`')
- each message is a series of fields separated by spaces
- **two types of messages**: "Add" and "Reduce" messages.
- Here's an example, which contains an "Add" message followed by a "Reduce" message:

```
A c S 97 36
R a 50
```

Add messages

```
'A' oid side price size
```

- '`A`': fixed string that identifies this as an "Add" message
- `oid`: "order id" used by subsequent "Reduce" messages
- `side`: '`B`' for a bid, '`S`' for an ask
- `price`: limit price of this order
- `size`: size of this order

Reduce messages

```
'R' oid size
```

- '`R`': fixed string identifying this as a "Reduce" message
- `oid`: "order id" identifies the order to be reduced
- `size`: amount by which to reduce the size of the order (*not* the new size of the order); if `size` is equal to or greater than the existing size of the order, the order is removed from the book

Processing messages

- "Reduce" messages affect at most one existing limit order
- "Add" messages either:
    - **add a single row to the book** (orders at the same price are stored separately to preserve "oid"s)
    - **cross the spread** and then (partially) remove any number of orders on the other side of the book (and may result in a new limit order of unmatched volume)
- Example message files are split into cases that include crosses and those that don't to help you develop your code incrementally and test it on inputs of differing difficulty

Ex: initial book

```
$ Rscript main.R solution.R input/book_1.csv input/empty.txt
$ask
  oid price size
1   a   105  100

$bid
  oid price size
1   b    95  100

Total volume: 100 100
Best prices: 95 105
Mid-price: 100
Spread: 10
```

Ex: processing a reduce message

```
$ cat input/message_ex_reduce.txt
R a 50
```

```
$ Rscript main.R solution.R input/book_1.csv input/message_ex_reduce.txt
$ask
  oid price size
1   a   105   50

$bid
  oid price size
1   b    95  100

Total volume: 100 50
Best prices: 95 105
Mid-price: 100
Spread: 10
```

Ex: Non-crossing add message

```
$ cat input/message_ex_add.txt
A c S 97 36
```

```
$ Rscript main.R solution.R input/book_1.csv input/message_ex_add.txt
$ask
  oid price size
2   a   105  100
1   c    97   36

$bid
  oid price size
1   b    95  100

Total volume: 100 136
Best prices: 95 97
Mid-price: 96
Spread: 2
```

Ex: crossing add message

```
$ cat input/message_ex_cross.txt
A c B 106 101
```

```
$ Rscript main.R solution.R input/book_1.csv input/message_ex_cross.txt
$ask
[1] oid   price size
<0 rows> (or 0-length row.names)

$bid
  oid price size
1   c   106    1
2   b    95  100

Total volume: 101 0
Best prices: 106 NA
Mid-price: NA
Spread: NA
```

9 longer sample output files

	messages_a	messages_ar	messages_arc
book_1			
book_2			
book_3			

```
output
■■■ book_1-message_a.out
■■■ book_1-message_ar.out
■■■ book_1-message_arc.out
■■■ book_2-message_a.out
■■■ book_2-message_ar.out
■■■ book_2-message_arc.out
■■■ book_3-message_a.out
■■■ book_3-message_ar.out
■■■ book_3-message_arc.out

0 directories, 9 files
```

Ex: two orders at the same price

Recall initial book:

```
$ Rscript main.R solution.R input/book_1.csv input/empty.txt
$ask
  oid price size
1   a   105  100

$bid
  oid price size
1   b    95  100

Total volume: 100 100
Best prices: 95 105
Mid-price: 100
Spread: 10
```

Ex: message_same_price.txt

```
$ Rscript main.R solution.R input/book_1.csv input/message_ex_same_price.txt
$ask
  oid price size
2   j   105  132
1   a   105  100

$bid
  oid price size
1   b    95  100
2   k    95   71

Total volume: 171 232
Best prices: 95 105
Mid-price: 100
Spread: 10
```

Earlier messages closer to the top of the book

Price-time precedence

- Orders are executed according to price time precedence
- Best price first, but **when two orders have the same price, the earlier one is executed first**
- We provide `book.sort` that respects price-time precedence
- It relies on the fact that the order ids increase as follows:

  ```
  a < k < ab < ba
  ```

  where < is indicating **"comes before"** in the message files

book.sort (in common.R)

```r
book.sort <- function(book, sort_bid=T, sort_ask=T) {
    if (sort_ask && nrow(book$ask) >= 1) {
        book$ask <- book$ask[order(book$ask$price,
                                   nchar(book$ask$oid),
                                   book$ask$oid,
                                   decreasing=F),]
        row.names(book$ask) <- 1:nrow(book$ask)
    }

    if (sort_bid && nrow(book$bid) >= 1) {
        book$bid <- book$bid[order(-book$bid$price,
                                   nchar(book$bid$oid),
                                   book$bid$oid,
                                   decreasing=F),]
        row.names(book$bid) <- 1:nrow(book$bid)
    }

    book
}
```

You are welcome (and encouraged) to use book.sort

Example output

```
$ask
  oid price size
8   a   105  100
7   o   104  292
6   r   102  194
5   k    99   71
4   q    98  166
3   m    98   88
2   j    97  132
1   n    96  375

$bid
  oid price size
1   b    95  100
2   l    95   29
3   p    94   87
4   s    91  102

Total volume: 318 1418
Best prices: 95 96
Mid-price: 95.5
Spread: 1
```

The rownames are now: 1,2,... starting from the best prices

book.summarise

```r
book.summarise <- function(book, with_stats=T) {
    if (nrow(book$ask) > 0)
        book$ask <- book$ask[nrow(book$ask):1,]

    book$ask <- book$ask[, c("oid", "price", "size")]
    book$bid <- book$bid[, c("oid", "price", "size")]

    print(book)

    if (with_stats) {
        clean <- function(x) { ifelse(is.infinite(x), NA, x) }

        total_volumes <- book.total_volumes(book)
        best_prices <- lapply(book.best_prices(book), clean)
        midprice <- clean(book.midprice(book))
        spread <- clean(book.spread(book))

        cat("Total volume:", total_volumes$bid, total_volumes$ask, "\n")
        cat("Best prices:", best_prices$bid, best_prices$ask, "\n")
        cat("Mid-price:", midprice, "\n")
        cat("Spread:", spread, "\n")
    }
}
```

Hints

For `book.spread` and `book.midprice` a nice implementation would use `book.best_prices`, which you should then implement first.

Turn on logging to help with debugging

```
Rscript main.R --log solution.R input/book_1.csv input/message_arc.txt
```

Then `book.reconstruct` uses `book.summarise` at every step

Remember to use `stringsAsFactors=FALSE` (e.g. for creating *data.frames*)

Hint for book.add & book.reduce

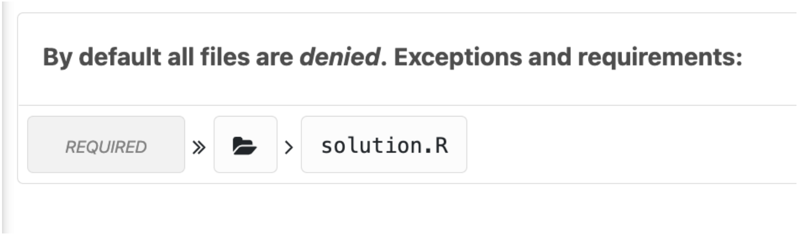
A possible way to implement `book.add` and `book.reduce` that makes use of the different example message files is the following:

1. Do a **partial implementation of book.add for messages that do not cross**, and check your implementation with `message_a.txt`.

2. **Implement book.reduce fully**. Check combined (partial) implementation of `book.add` and `book.reduce` with `message_ar.txt` and `book_3.csv` (only this combination with `message_ar.txt` has no crosses).

3. **Complete book.add implementation**. Check crosses using `message_arc.txt` and any initial book, or with `message_ar.txt` and `book_1.csv` or `book_2.csv`

Submit only "solution.R"


By default all files are *denied*. Exceptions and requirements:

REQUIRED » 📂 › solution.R

Plagiarism / collusion

Warning

- **Do not show your work to other students or search for solutions online**.
- Automatic plagiarism/collusion detection is in place, and **suspected cases are passed on to the academic integrity officer**.
- **Students found to have plagiarized or colluded get 0**, with further sanctions such as termination of studies for repeat offences.

4 "extra" functions to implement

Final 30% is only available if CodeGrade gives you full marks (70%) for the pre-deadline tests.

Only focus on the extra problems once you have achieved this.

You can get marks for any one of these independently:

1. `book.extra1 <- function(book,size)` **[6%]**
2. `book.extra2 <- function(book,size)` **[6%]**
3. `book.extra3 <- function(book)` **[6%]**
4. `book.extra4 <- function(book,k)` **[12%]**

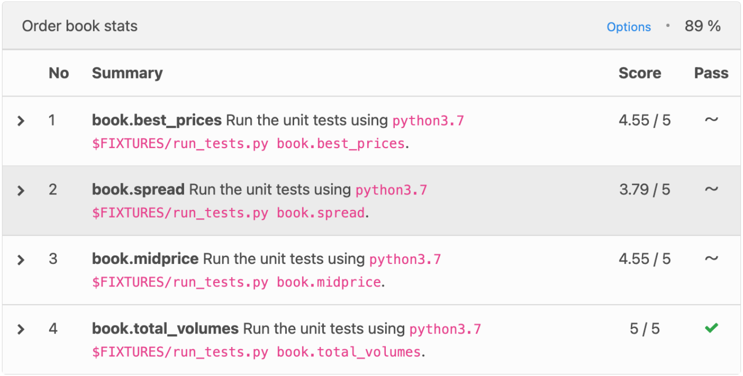
Using CodeGrade to improve

After you submit the tests for the first 6 functions are run. After a short while you get:

1. a **provisional mark** is visible;
2. to **see any tests that you failed**, with details of the inputs, the expected, and the received output.

To illustrate, I edited the correct solution so that `book.best_prices` give the wrong answer 10% of the time at random, and submitted this as a test student:

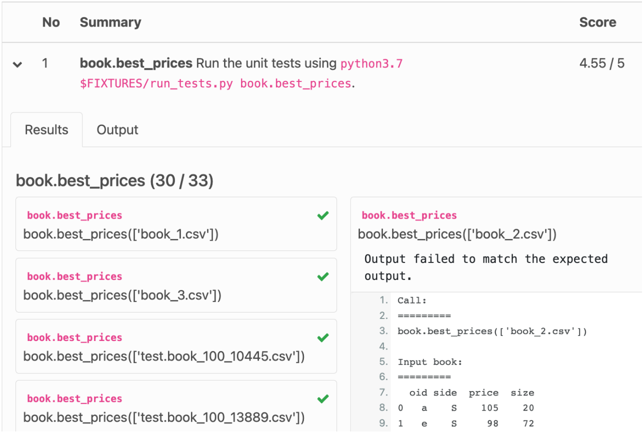
Example: Order book stats



Order book stats   Options • 89 %

	No	Summary	Score	Pass
›	1	**book.best_prices** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.best_prices`.	4.55 / 5	~
›	2	**book.spread** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.spread`.	3.79 / 5	~
›	3	**book.midprice** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.midprice`.	4.55 / 5	~
›	4	**book.total_volumes** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.total_volumes`.	5 / 5	✓

Example: book.best_prices



No	Summary	Score
1	**book.best_prices** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.best_prices`.	4.55 / 5

Results Output

book.best_prices (30 / 33)

`book.best_prices`
book.best_prices(['book_1.csv']) ✓

`book.best_prices`
book.best_prices(['book_3.csv']) ✓

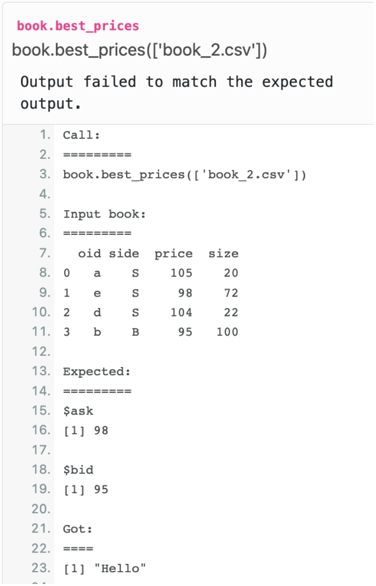
`book.best_prices`
book.best_prices(['test.book_100_10445.csv']) ✓

`book.best_prices`
book.best_prices(['test.book_100_13889.csv']) ✓

`book.best_prices`
book.best_prices(['book_2.csv'])

```
Output failed to match the expected
output.
```

```
Call:
=========
book.best_prices(['book_2.csv'])

Input book:
=========
  oid side  price  size
0   a    S    105    20
1   e    S     98    72
```

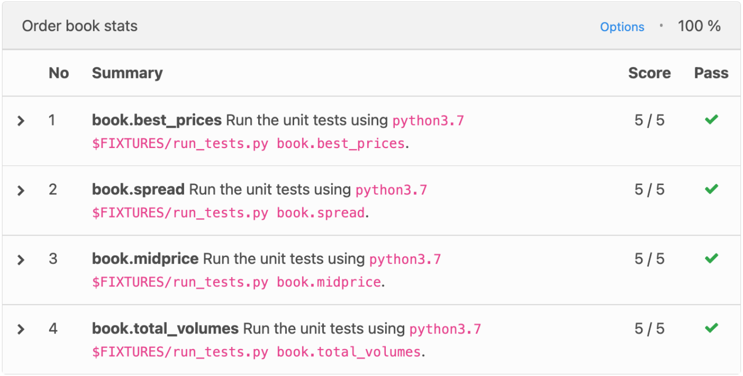

book.best_prices

book.best_prices(['book_2.csv'])

```
Output failed to match the expected
output.
```

```
Call:
=========
book.best_prices(['book_2.csv'])

Input book:
=========
  oid side  price  size
0   a    S    105    20
1   e    S     98    72
2   d    S    104    22
3   b    B     95   100

Expected:
=========
$ask
[1] 98

$bid
[1] 95

Got:
====
[1] "Hello"
```

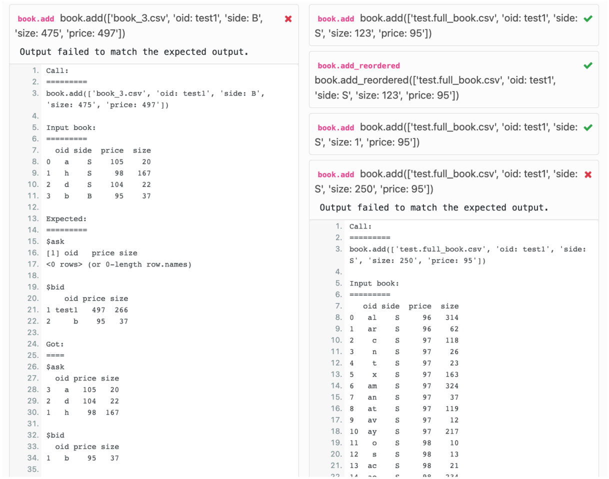
Example: correct order book stats



Order book stats Options • 100 %

	No	Summary	Score	Pass
›	1	**book.best_prices** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.best_prices`.	5 / 5	✓
›	2	**book.spread** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.spread`.	5 / 5	✓
›	3	**book.midprice** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.midprice`.	5 / 5	✓
›	4	**book.total_volumes** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.total_volumes`.	5 / 5	✓

Example test for book.add



book.add book.add(['book_3.csv', 'oid: test1', 'side: B', 'size: 475', 'price: 497']) ✗

```
Output failed to match the expected output.
```

```
Call:
=========
book.add(['book_3.csv', 'oid: test1', 'side: B',
'size: 475', 'price: 497'])

Input book:
=========
  oid side  price  size
0   a    S    105    20
1   h    S     98   167
2   d    S    104    22
3   b    B     95    37

Expected:
=========
$ask
[1] oid   price size
<0 rows> (or 0-length row.names)

$bid
    oid price size
1 test1   497  266
2     b    95   37

Got:
====
$ask
  oid price size
3   a   105   20
2   d   104   22
1   h    98  167

$bid
  oid price size
1   b    95   37

```

book.add book.add(['test.full_book.csv', 'oid: test1', 'side: S', 'size: 123', 'price: 95']) ✓

book.add_reordered book.add_reordered(['test.full_book.csv', 'oid: test1', 'side: S', 'size: 123', 'price: 95']) ✓

book.add book.add(['test.full_book.csv', 'oid: test1', 'side: S', 'size: 1', 'price: 95']) ✓

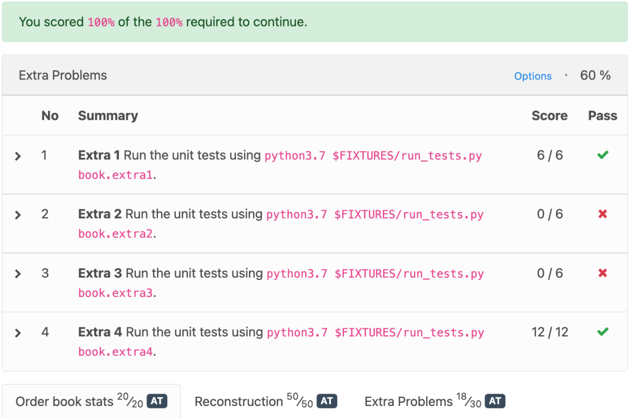
book.add book.add(['test.full_book.csv', 'oid: test1', 'side: S', 'size: 250', 'price: 95']) ✗

```
Output failed to match the expected output.
```

```
Call:
=========
book.add(['test.full_book.csv', 'oid: test1', 'side:
S', 'size: 250', 'price: 95'])

Input book:
=========
   oid side  price  size
0   al    S     96   314
1   ar    S     96    62
2    c    S     97   118
3    n    S     97    26
4    t    S     97    23
5    x    S     97   163
6   am    S     97   324
7   an    S     97    37
8   at    S     97   119
9   av    S     97    12
10  ay    S     97   217
11   o    S     98    10
12   s    S     98    13
13  ac    S     98    21
```

Tests for extra functions

Similar test output for extra functions, provided that you got the first 6 functions totally correct, **but only after the deadline**:



You scored 100% of the 100% required to continue.

Extra Problems — Options · 60 %

No	Summary	Score	Pass
1	**Extra 1** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.extra1`.	6 / 6	✓
2	**Extra 2** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.extra2`.	0 / 6	✗
3	**Extra 3** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.extra3`.	0 / 6	✗
4	**Extra 4** Run the unit tests using `python3.7 $FIXTURES/run_tests.py book.extra4`.	12 / 12	✓

Order book stats 20/20 AT   Reconstruction 50/50 AT   Extra Problems 18/30 AT

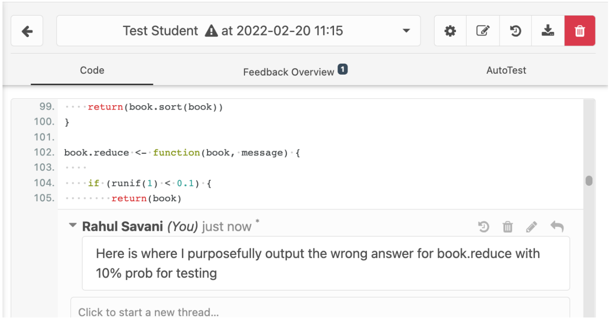
Tests

- **First few tests** in each category **use book_{1,2,3}.csv**.
- Many tests use more complicated csv files, and while the data is visible in CodeGrade, we do not give you the csv file.
- **Some tests are randomly generated** (needed to disincentivise hardcoding).

  This means that you may experience a small amount of variance in the mark you see when you resubmit the same wrong code (correct code will get full marks every time).

Submit to CodeGrade for help

- we can see test results, which helps us to help you;
- we can enter comments directly on submitted source code:



```
    return(book.sort(book))
}

book.reduce <- function(book, message) {
    
    if (runif(1) < 0.1) {
        return(book)
```

Rahul Savani *(You)* just now *

Here is where I purposefully output the wrong answer for book.reduce with 10% prob for testing

Click to start a new thread...