

Session 18

Console expense splitter built from the Session 17 study, with sample run and comparison against the original

This session turns the Session 17 study into a working program. The deliverable is `expense_splitter.py`, a console program that adds members, records expenses with equal or custom shares, shows balances and prints a minimal settlement plan, followed by a real sample run, a comparison of the original app's behaviour with this program, and notes on what the exercise taught about reverse engineering. Everything the program does traces back to a rule or requirement written in Session 17; nothing was added that the original did not show.

Objectives

✗ Do not copy. Read for understanding and the viva

- Implement requirements ES-FR-1 to ES-FR-7 from Session 17 in one readable Python file.
- Reproduce the observed rounding and settlement behaviour exactly.
- Compare the rebuilt portion with the original feature by feature and record the differences.
- Reflect on which inferences from Session 17 held and which had to be revised.

Problem Statement

■ Write in lab record

Sessions 17 and 18: Have a look at the output of any program that was not written by you. Preferably, look at an application that is not developed by you and write the program for the development of that application or a portion of that application.

Concept

✗ Do not copy. Read for understanding and the viva

From inferred rules to code

Each rule in Session 17 becomes one small function. R2 (equal split with leftover paise to the first members) is `equal_split`. R4 (net balance = paid minus owed) is `balances`. R6 (greedy settlement) is `settle`. When a rule maps to one function, the comparison table at the end can point at the exact place where behaviour matched or differed.

Money as integers

Rupees with two decimals are stored as an integer number of paise. `to_paise("33.33")` gives 3333 and `fmt(3333)` gives back "33.33". Division then uses `divmod`, which returns the equal share and the leftover paise in one step, and the leftover is handed out one paisa each to the first members. This reproduces the observed 33.34, 33.33, 33.33 with no floating point at all.

Greedy settlement

Sort debtors by how much they owe (largest first) and creditors by how much they are owed (largest first). Pay the largest creditor from the largest debtor with the smaller of the two amounts; one of them reaches zero and drops out; repeat. Every payment retires at least one member, so a group of n members with non-zero balances needs at most n minus 1 payments. This matches what the original app showed for the test group, although the original may pair members in a different order.

What is left out on purpose

The original stores data on a server and syncs between phones; this program keeps everything in memory for one run. The original supports percentages and share weights; this program supports equal and exact amounts only, which is what Session 17 recorded as the portion. Leaving these out is a decision, and the comparison table records it.

Program

■ Write in lab record

expense_splitter.py

```

1  """Expense Splitter: a console re-implementation of the group expense
2  portion of a bill-splitting app (Splitwise style), written from observed
3  behaviour only. Python 3, standard library, no persistence.
4
5  All money is kept in paise (integers) so equal splits round exactly the
6  way the original app does: the leftover paise go to the first members.
7  """
8
9
10 def to_paise(text):
11     """Convert a rupee string such as '1200.50' to an integer of paise."""
12     rupees, _, paise = text.strip().partition(".")
13     paise = (paise + "00")[:2]
14     return int(rupees or 0) * 100 + int(paise)
15
16
17 def fmt(paise):
18     """Format paise as a rupee string, e.g. 120050 → '1200.50'."""
19     sign = "-" if paise < 0 else ""
20     paise = abs(paise)
21     return f"{sign}{paise // 100}.{paise % 100:02d}"
22
23
24 def equal_split(total, names):
25     """Split total paise equally; remainder paise go to the first names."""
26     base, extra = divmod(total, len(names))
27     return {n: base + (1 if i < extra else 0) for i, n in enumerate(names)}
28
29
30 def balances(members, expenses):
31     """Net balance per member: positive = is owed, negative = owes."""
32     net = {m: 0 for m in members}
33     for e in expenses:
34         net[e["paid_by"]] += e["amount"]
35         for name, share in e["shares"].items():
36             net[name] -= share
37     return net
38
39
40 def settle(net):
41     """Greedy settlement: repeatedly pay the largest creditor from the
42     largest debtor. Returns a list of (from, to, paise) transactions."""
43     debtors = sorted(((v, k) for k, v in net.items() if v < 0))

```

```
44     creditors = sorted(((v, k) for k, v in net.items() if v > 0), reverse=True)
45     result = []
46     i = j = 0
47     while i < len(debtors) and j < len(creditors):
48         owed, debtor = debtors[i]
49         due, creditor = creditors[j]
50         pay = min(-owed, due)
51         result.append((debtor, creditor, pay))
52         debtors[i] = (owed + pay, debtor)
53         creditors[j] = (due - pay, creditor)
54         if debtors[i][0] == 0:
55             i += 1
56         if creditors[j][0] == 0:
57             j += 1
58     return result
59
60
61 def pick_member(members, prompt):
62     """Ask for a member name until it matches one in the group."""
63     while True:
64         name = input(prompt).strip()
65         if name in members:
66             return name
67         print(f" no such member: {name}. Members: {' '.join(members)}")
68
69
70 def add_member(members):
71     name = input("Member name: ").strip()
72     if not name or name in members:
73         print(" name is empty or already in the group")
74         return
75     members.append(name)
76     print(f" added {name}")
77
78
79 def add_expense(members, expenses):
80     if len(members) < 2:
81         print(" add at least two members first")
82         return
83     desc = input("Description: ").strip() or "expense"
84     amount = to_paise(input("Amount (rupees): "))
85     if amount <= 0:
86         print(" amount must be positive")
87     return
```

```
88     paid_by = pick_member(members, "Paid by: ")
89     mode = input("Split equally (e) or custom shares (c)? ").strip().lower()
90     if mode == "c":
91         shares = {}
92         for m in members:
93             shares[m] = to_paise(input(f" share of {m}: ") or "0")
94         if sum(shares.values()) != amount:
95             print(f" shares total {fmt(sum(shares.values()))}, "
96                   f"expense is {fmt(amount)}. Expense not added.")
97             return
98         shares = {k: v for k, v in shares.items() if v}
99     else:
100         shares = equal_split(amount, members)
101
102     expenses.append({"desc": desc, "amount": amount,
103                    "paid_by": paid_by, "shares": shares})
104
105     print(f" added '{desc}' {fmt(amount)} paid by {paid_by}")
106
107 4
108 10
109 5
110 10
111 6 def show_expenses(expenses):
112 10
113 7     if not expenses:
114 10
115 8         print(" no expenses yet")
116 10
117 9     for n, e in enumerate(expenses, 1):
118 11
119 0         parts = ", ".join(f"{k} {fmt(v)}" for k, v in e["shares"].items())
120 11
121 1         print(f" {n}. {e['desc']:<14} {fmt(e['amount']):>9} "
122 11
123 2                 f"paid by {e['paid_by']:<8} shares: {parts}")
124 11
125 3
126 11
127 4
128 11
129 5 def show_balances(members, expenses):
```

```

11         for name, net in balances(members, expenses).items():
12
13             if net > 0:
14
15                 print(f" {name:<8} gets back {fmt(net)}")
16
17             elif net < 0:
18
19                 print(f" {name:<8} owes {fmt(-net)}")
20
21             else:
22
23                 print(f" {name:<8} settled up")
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1
```

```
13
8
13
9
14
0 def main():
14
1     members, expenses = [], []
14
2     actions = {"1": lambda: add_member(members),
14
3                 "2": lambda: add_expense(members, expenses),
14
4                 "3": lambda: show_expenses(expenses),
14
5                 "4": lambda: show_balances(members, expenses),
14
6                 "5": lambda: show_settlement(members, expenses)}
14
7 while True:
14
8     print(MENU)
14
9     try:
15
0         choice = input("Choice: ").strip()
15
1     except EOFError:
15
2         break
15
3     if choice == "0":
15
4         break
15
5     actions.get(choice, lambda: print(" unknown choice"))()
15
6 print("Bye")
15
7
15
8
```

```
15
9  if __name__ == "__main__":
16
0      main()
```

Sample Output

The run below was produced by piping the following choices into the program, so each prompt and the typed value appear on one line. Input given, in order: add members Asha, Bimal, Chetan, Divya; expense Dinner 1000 paid by Asha split equally; expense Cab 350 paid by Bimal with custom shares 100, 100, 150, 0; expense Tickets 800 paid by Chetan split equally; then list expenses, balances, settle up, exit.

```

1 1 Add member    2 Add expense    3 List expenses
2 4 Balances      5 Settle up      0 Exit
3
4 Choice: Member name:    added Asha
5
6 1 Add member    2 Add expense    3 List expenses
7 4 Balances      5 Settle up      0 Exit
8
9 Choice: Member name:    added Bimal
10
11 1 Add member   2 Add expense   3 List expenses
12 4 Balances     5 Settle up     0 Exit
13
14 Choice: Member name:    added Chetan
15
16 1 Add member   2 Add expense   3 List expenses
17 4 Balances     5 Settle up     0 Exit
18
19 Choice: Member name:    added Divya
20
21 1 Add member   2 Add expense   3 List expenses
22 4 Balances     5 Settle up     0 Exit
23
24 Choice: Description: Amount (rupees): Paid by: Split equally (e) or custom s
25
26 1 Add member   2 Add expense   3 List expenses
27 4 Balances     5 Settle up     0 Exit
28
29 Choice: Description: Amount (rupees): Paid by: Split equally (e) or custom s
30
31 1 Add member   2 Add expense   3 List expenses
32 4 Balances     5 Settle up     0 Exit
33
34 Choice: Description: Amount (rupees): Paid by: Split equally (e) or custom s
35
36 1 Add member   2 Add expense   3 List expenses
37 4 Balances     5 Settle up     0 Exit
38
39 Choice:    1. Dinner                1000.00  paid by Asha      shares: Asha 250.00,
40           2. Cab                    350.00  paid by Bimal    shares: Asha 100.00, Bimal 1
41           3. Tickets                 800.00  paid by Chetan   shares: Asha 200.00, Bimal 2
42
43 1 Add member   2 Add expense   3 List expenses

```

```

44 4 Balances      5 Settle up      0 Exit
45
46 Choice:  Asha      gets back 450.00
47   Bimal    owes      200.00
48   Chetan   gets back 200.00
49   Divya    owes      450.00
50
51 1 Add member    2 Add expense    3 List expenses
52 4 Balances      5 Settle up      0 Exit
53
54 Choice:   Divya pays Asha 450.00
55   Bimal pays Chetan 200.00
56   (2 transaction(s))
57
58 1 Add member    2 Add expense    3 List expenses
59 4 Balances      5 Settle up      0 Exit
60
61 Choice: Bye

```

Hand check of the balances: Asha paid 1000 and owes $250 + 100 + 200 = 550$, net +450. Bimal paid 350 and owes 550, net -200. Chetan paid 800 and owes $250 + 150 + 200 = 600$, net +200. Divya paid nothing and owes 450. The four nets sum to zero (rule R5). Without simplification the group has four pairwise debts (Divya to Asha, Divya to Chetan, Bimal to Asha, Bimal to Chetan); the plan settles it in two.

The rounding case from Session 17, three members and an expense of 100 split equally, produced this line in a second run:

```

1    1. Tea                100.00  paid by A        shares: A 33.34, B 33.33, C

```

Comparison with the Original

■ Write in lab record

Feature	Original application (observed)	This program	Match
Add member	By name or email; duplicates refused	By name; blank and duplicate refused	Partial: no email
Add expense	Description, amount, payer, split; date automatic	Same fields; no date	Partial: no date
Equal split	1000 among 4 gives 250.00 each	250.00 each	Yes
Rounding	100 among 3 gives 33.34, 33.33, 33.33	33.34, 33.33, 33.33	Yes
Custom split	Exact amounts, percentages, shares; must total the amount	Exact amounts only; refused if total differs	Partial: one mode
Exclude a member	Untick the member	Give the member a share of 0; share not listed	Yes, different input
Balances	"owes", "gets back", "settled up" with colours	Same three words, no colours	Yes
Settle up	Two payments for the test group with simplify debts on	Two payments, same pairs	Yes
Expense list	Date order, shows viewer's share	Entry order, shows all shares	Partial
Edit or delete expense	Supported	Not supported	No
Persistence	Server, multi-device	None; in memory for one run	No, out of scope
Interface	Touch screen forms	Numbered console menu	No, by design

Rules R1 to R5 and R7 were reproduced exactly. R6 produced the same number of payments and the same pairs for the test group, so the inference stands for this case; it has not been shown to hold for every case.

What Reverse Engineering Taught

■ Write in lab record

- The observable rounding rule was enough to fix the internal representation: integer paise. One observation settled a design decision.
- The behaviour of the settlement screen could be matched without knowing the original algorithm. Two different algorithms can be indistinguishable from outside; a black-box study only proves behaviour on the cases tried.
- The custom-share check (shares must total the amount) was found only because the original's save button refused to enable. Error behaviour is as much a requirement as normal behaviour, and it is easy to miss when only success paths are tried.
- Writing the requirements as ES-FR ids before coding made the comparison table mechanical: one row per requirement, match or not.
- Deciding the portion up front kept the rebuild to 160 lines. Every "No" in the comparison table is a scope decision made in Session 17, not a failure discovered in Session 18.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** How do you know your program matches the original? **A:** By the comparison table: the same inputs were given to both and the outputs are listed side by side.
- **Q:** Why does `equal_split` use `divmod`? **A:** It returns the base share and the leftover paise together; the leftover is distributed one paisa each to the first members.
- **Q:** What happens if custom shares do not total the amount? **A:** The expense is rejected with a message showing both totals, as the original refuses to save.
- **Q:** Why sort debtors and creditors before settling? **A:** So that the largest debt is retired against the largest credit first, which keeps the payment count small.
- **Q:** What is the maximum number of payments the greedy method produces? **A:** One fewer than the number of members with a non-zero balance.

- **Q:** Which inferred rule is least certain and why? **A:** R6, because different pairing orders give the same payment count and cannot be told apart from the screen.
- **Q:** Why is data not saved to a file? **A:** Persistence was outside the portion chosen in Session 17; adding it would not change the rules being studied.
- **Q:** What would you test next to strengthen the R6 inference? **A:** Groups where the greedy method is known not to be minimal, and check whether the original does better.

Common Mistakes

✖ Do not copy. Read for understanding and the viva

- Using floating point for money and getting shares that total 99.99 or 100.01.
- Building features the original has but Session 17 never recorded, so there is nothing to compare them against.
- Pasting output typed by hand instead of captured from a real run; examiners try the program.
- A comparison table with only "Yes" rows; honest "Partial" and "No" rows show the scope was understood.
- Letting a member both pay and receive in the settlement plan, which means balances were not netted first.

Session Summary

■ Write in lab record

- Source listing of `expense_splitter.py` with the function-to-rule mapping noted in the margin.
- Sample run with the same numbers used on the original application, plus the rounding case.
- Hand check that balances sum to zero and the settlement plan clears every balance.
- Comparison table, one row per observed feature.
- Notes on what reverse engineering taught, including which inference is still uncertain.

[✎ Edit this page](#)

< Previous
Session 17

Next >
Session 19